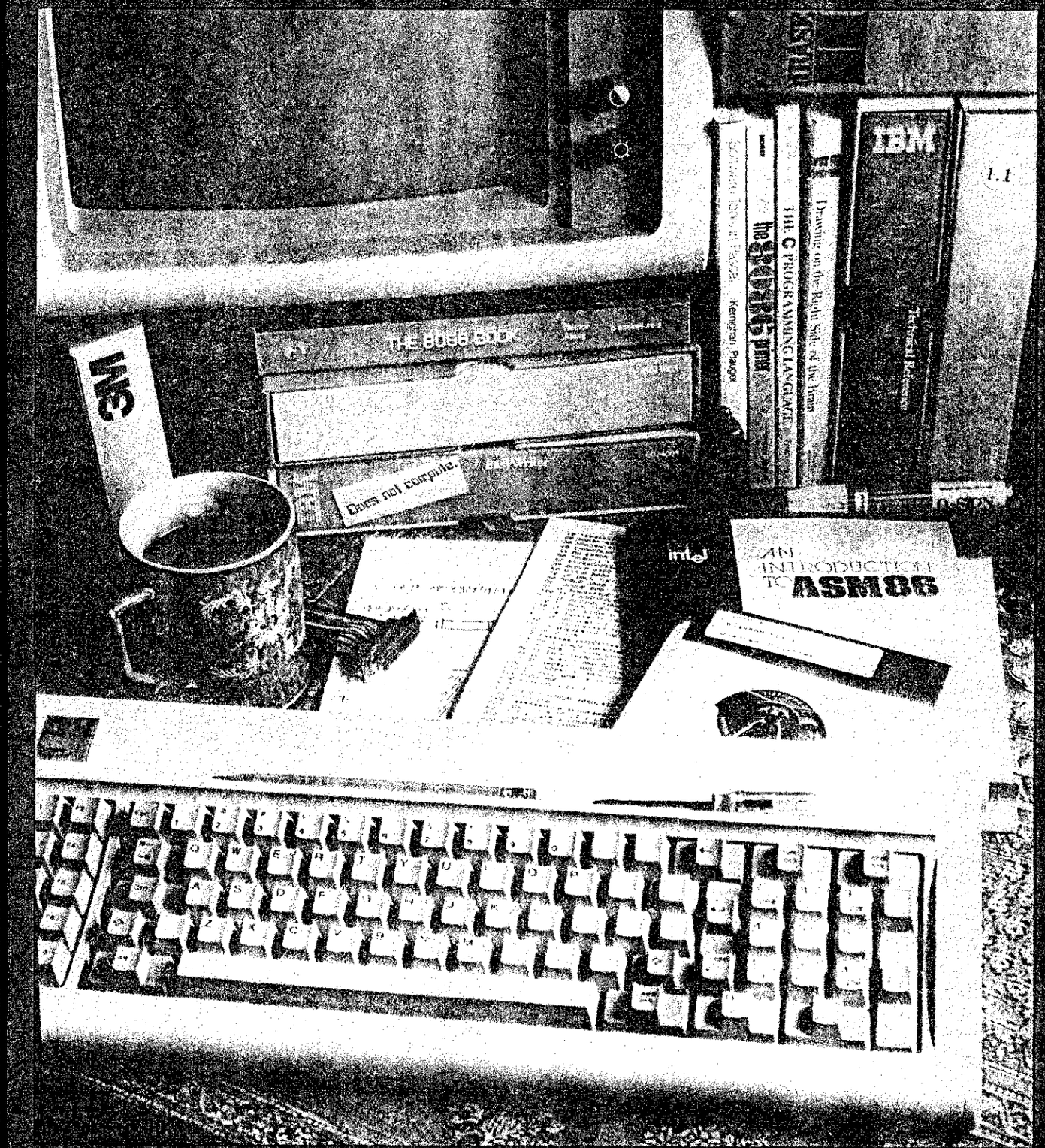
PROGRAMMER'S JOURNAL

The Resource Journal for IBM PC Programmers



Volume 1

January 1983

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FEATURES:

STYLE MANUAL / Ted Lewis 3
Your guide to user friendly software

PJ PROFILE / Interview 5
A conversation with the author of the Norton Utilities

BATCH FILES & PASCAL / Peter Norton 7
Create efficient batch files and do Pascal I/O

COBOL / Rich Rohde 11
An evaluation of IBM's Cobol

SYSTEM ANALYSIS / with Bruce Huntington 20
An orthodox approach to system design

ROUNDING / D. Jones 22
Take a look at the many aspects of rounding error

DEPARTMENTS:

EDITORIAL / LETTERS 2
FORUM / Frankie Detesto 13
TOOLS OF THE TRADE / BASIC 14
C 17
dBASE II 18
UTILITIES 24
SOFTWARE HOUSES 25
ASM 27
RESOURCE LIBRARY 28
MISC. 29

Editor/Publisher: Greg Estes
Associate Editor: Frankie Detesto
Contributing Editor: Greg Roberts
Art Director: Julia King
Production: Cassandra Estes
Programming Consultants
Rich Rohde
D. Jones
Les Moore
Peter Callahan
Bruce Huntington

Editorial and Business Office: 2765 Potter St., Eugene, OR 97405

Mailing Address: P.O. Box 5070, Eugene, OR 97405

Programmer's Journal. Volume 1, January. Copyright (C) 1983 by Programmer's Journal. Published bi-monthly.

Postmaster: Send address changes to Programmer's Journal, 2765 Potter St. Eugene, OR 97405
Ph. (503)687-0398

Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada, and $40 elswhere.)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York
dBASEII is a trademark of Ashton-Tate.

Special thanks to Les and Sandy Moore for their help and continued support.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR

All of the articles in Programmer's Journal are written by other programmers with a deep commitment to sharing what they have learned. The readers of this publication are also the authors of it. As we become more experienced on the IBM PC the demands on our time also increase, but many programmers are still willing to spend the extra effort to pass on their hard won knowledge. For no other reason than to continue the gift of time and energy that someone once gave them.

In one sense PJ is a national communication link for IBM programmers who know that dialogue is an essential key in developing quality software.

Learning to improve your programming skills on a personal computer is a very difficult process to say the least. If we all began our home computer careers by working for Microsoft for 3 months, then transfering to IBM's Boca Raton headquarters for 6 months, and finished up with a quick stint at Intel we would be in excellent shape to return home and quietly pursue our own software careers. Even a programmer who works in a mainframe shop need not be frustrated for more than a few minutes with a difficult software problem. Just by turning to the senior analyst next to them they can often have their questions answered.

Personal computer programming is an entirely different matter. Our tools consist of a computer, some reference books and a few magazines with hopefully a column that occasionally sheds some light on our seemingly endless series of road blocks. If we happened to have held a professional programming position at some time in our life we are indeed truly lucky. But then the burden of sharing falls more on our shoulders, and not everyone is willing to shoulder that task.

The IBM PC has taken the personal computer into a new dimension. The contributors to Programmer's Journal are working to do the same for software

—Greg Estes

LETTERS

A PIECE OF THE ACTION

Thanks for your letter. I'm enclosing a copy of the Character Definition Set for your review. Your Journal sounds interesting, too. Would you be interested in an article on how to implement user-defined graphics? Let me know.

In any case, I'm also enclosing my subscription order. ($24 for 6 issues? That's more than PC!)

Mark Minasi
Arlington, VA

(Mark let us know of a new bug in DOS/BASIC. If you do a FORMAT B:/S at some point and then do a BSAVE in a BASIC program you get a "Syntax Error" in the BSAVE line and the program hangs. IBM acknowledged the error and may fix it in the future. The error does not occur if a BSAVE is done in interpretive mode.)

N.Y.P.C.

Your (journal) idea is downright brilliant. I will be even more impressed when I see that you really have figured out how to get programmers to write about their work in English.

Send your flyers. We'll be glad to hand them out at our meeting.

Joe Rigo
NY NY

YOU FIRST

We will (send a copy of our newsletter,) just as soon as you send US a copy of YOUR publication, so that we may examine it. Right now it's a Pig in the Poke.

Christian Du Lac
Bay Area Users Group
San Francisco, CA

PROGRAMMERS

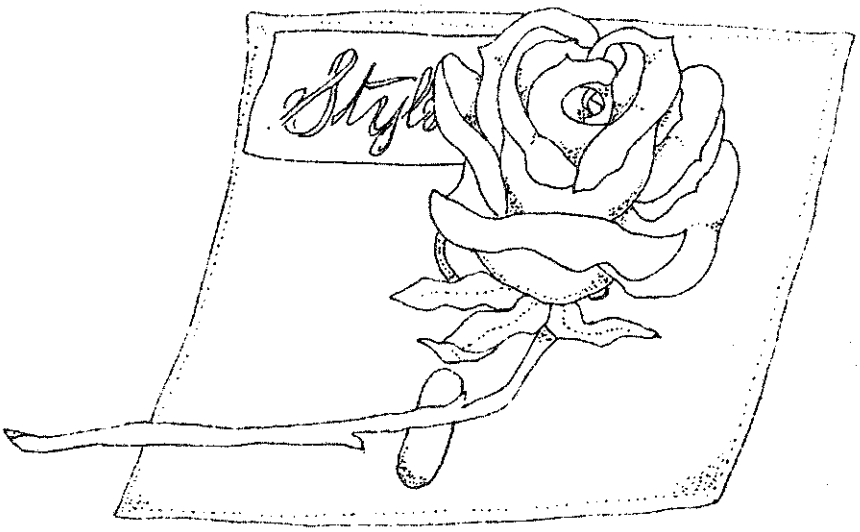
Your ideas, opinions, words of advice and pieces of code are the most important part of Programmer's Journal. The lessons we learn while working on our IBM PC's are appreciated most of all by our fellow programmers.

Articles, source code, and lengthly letters should be sent on diskette. Your disks will be returned. They should be in standard ASCII format. (Wordstar users should edit with a "Non-document" option.)

Enclose hard-copy also.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STYLE



A Style Manual for Software Authors

by Ted Lewis

(The following recommendations are excerpts from a style manual developed by Ted Lewis. They are designed to cover many of the features found in "good" software.)

General Program Guidelines

1. Initially clear the screen so that remnants of other programs are not visible when the user begins.

2. Set the mode of the computer as needed. For example, if you are going to use graphics and uppercase letters the program should set them accordingly. No assumptions should be made about what a previous program has done.

3. Print the name of the program along with a greeting message if appropriate so the user can verify which program is running.

4. Ask the user if they would like to see instructions. If the user is familiar with the program, they may want to skip them.

5. Use menus and autoload files so the user never has to resort to an operating system command or any other cryptic command in order to use your program.

6. Use a full screen format when displaying information instead of scrolling a single line at a time. Never use timed displays; instead, allow people to read at their own rates. You should:

   a. Clear the screen.
   b. Display at most one screen full of text to prevent scrolling.
   c. Save the last line of each page for displaying error messages or prompting messages such as "Press space bar to continue."
   d. Go to the next page when the space bar is pressed.
   e. Scroll only when necessary. For instance, when writing a story the user may want to see not only the last line they typed, but several preceeding lines.

7. Consistantly use the same key (e.g. space bar) to indicate when the user is ready to continue. Again, do not use timed delays.

8. Begin and end words on the same line. Do not hyphenate them or have them begin on one line and end on another (wraparound.)

9. Allow enough room to PRINT statements or variables without disturbing the screen or printer format.

10. Allow the user to display the instructions, stop the activity or restart the program at intervals throughout the program.

11. When the program is going to take some time to make calculations or fill an array, print a message on the screen asking the user to wait.

12. Allow the user to watch their own progress through an activity when appropriate. Highlighting or otherwise showing the completed sections.

13. Use graphics and sound whenever appropriate to make the program interesting. If the user may not want these features, they should be allowed to cancel them.

14. When the user finishes or interrupts the activity, have the program Clear the screen, Comment on user performance if appropriate, END, leaving comments on the screen and placing the cursor on a clear line so that a command can be entered or ask the user to press the Space bar and then run a menu program which lists the names of other programs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Forms and Prompts

— Use prompt lines in a screen format to capture input data, user responses, and control information. The screen format should appear all at once. Avoid scrolling; instead, move the cursor to the appropriate prompt line.
— Use short, descriptive screen lines. Avoid clever or cute phrases as well as computer terminology. Words such as FILE, BYTE, OVERFLOW, etc. should be avoided; in their place use INFORMATION, CHARACTER, TOO BIG, etc.
— Edit inputs before they are accepted by your program as described below under INPUT.

Instructions

1. Double space the text if it will be easier to read.
2. Limit instructions to three screen pages in length.
3. Use graphic illustrations whenever possible.
4. Have a key (e.g. 'B' for back) that allows the user to go back to the previous page of instructions.
5. After the instructions are printed, ask the user if they want to continue with the program or stop. They may not wish to proceed.
6. Include reminders throughout the program. For example, if the user is to enter a coordinate, the program might remind him to enter two numbers separated by a comma.

Help Screens

— Allow users to obtain brief explanations of what they are supposed to do next through a help screen.
— Help screens should be activated by pressing a single key.
— Make help screens available as frequently and easily as practical.
— Store help screen text on diskettes so they can be changed without changing the program. Also, this reduces RAM requirements.
— Use graphics and sound to enhance your program, but do not overwhelm the user.
— Do not obstruct the expert user's progress.

Menus

— Always use menu-selection as a means of capturing commands where possible.
— Structure menus so their hierarchy is readily understandable to the user. Make the 'big picture' as clear as possible. Document this structure in your manual.
— Do not overburden a menu. Use carefully selected phrases that succinctly tell the user what his selection will cause the computer to do. Try to keep the number of menu items to a minimum (less than 10 choices per menu.)
— Alert the user to any delays in processing that may occur.

INPUT

1. Use input routines rather than input statements. The BASIC input statement has several pitfalls which can be avoided. Your routine should:

a. Clear the keyboard buffer before receiving input. Previously typed characters may be mistakes, so they should be discarded.
b. Prevent the profeam from ending if the user presses the RETURN key without having entered any information.
c. Prevent the user from deleting too many characters when correcting their errors. (i.e. they should not be able to delete text from the screen format.)
d. Screen out characters that would clear the screen, home the cursor, or move the cursor to an unwanted position.
e. Limit the number of characters the user may input. For example, if you want to accept a maximum of nine characters, the user should not be able to enter more than nine.
f. Have the cursor move to the left and erase one character at a time when the user is deleting errors. Characters displayed to the right of the cursor should not be dragged to the left as the cursor moves.

2. Have the program wait for the user to press RETURN after his input, rather than going on with the program as soon as any key is pressed.
3. Check the input for errors. If errors are made, print an explanation of the error and then ask for the input again. For example, if a user enters a letter instead of a number, an error message might say, 'Remember, you can only use numbers.'
4. Allow enough room after a question (e.g. 'What is your name?') for the input to be typed without wrapping around to the next line.
5. For Yes/No questions, check only the first letter that is entered.
6. Check for errors the program might make as a result of user input (e.g. divide by zero, string overflow, etc.) Give explicit messages to the user to explain their mistake.

The above standards reflect an attempt to make programs self-explanatory, easy to understand, and relatively free of bugs. We welcome suggested additions to this list. □

For additional information, a copy of the 'Software Authors Guide', or details about software submissions, contact Ted Lewis, Software Editor, dilithium Press, 11000 SW 5t., Suite E, Beaverton, OR 97005

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

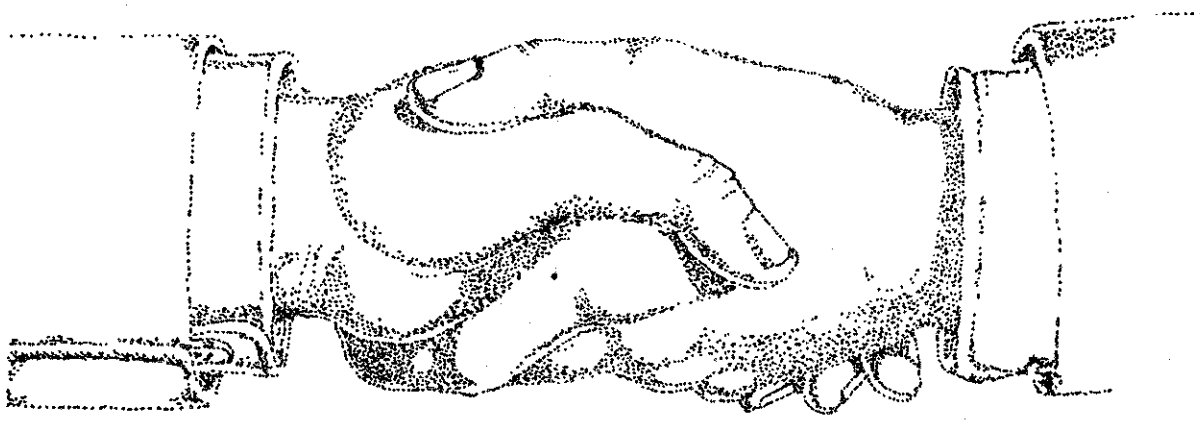
PJ PROFILE

Peter Norton and the Norton Utilities

(An interview with the author of the NORTON UTILITIES, a series of utility programs that visually uncover the mystic surrounding diskettes. Ideal tools for anyone that desires a full understanding of how information is stored and recovered on floppy disks.)

PJ: When did you first get your IBM PC?

PETER: I ordered my IBM PC in November (1981) but of course didn't get it until Febuary of 82, there was a long lead time during that period. All my previous experience had been with main frames. I had literally not even touched a micro until the IBM arrived. The excitement and advanced features of the PC convinced me to go ahead and purchase a home computer.



PJ: Did you have definate plans for developing software?

PETER: When it arrived I was still working full time on an assignment and had no specific plans about software. The only idea that occurred to me was doing a prime number sift if you can believe that. I really had no intended use for the machine at all. The Utility software did not evolve as a planned business venture, it was more of an accident than anything else. When I started playing with the PC there was a lot of information I wanted to know. For instance, what was going on inside of diskettes. That became the impulse behind writing DISKLOOK. It let me snoop around very thoroughly in disk files. Like many people I also inadvertantly erased an important program file and this encouraged me to write UNERASE. Having produced those 2 programs I realized that other IBM PC users would end up needing the same tools. The results turned into the Norton Utilities which were entirely created out of a need to solve the problems I faced with my own software development.

PJ: How long had you used Pascal?

PETER: My exposure to Pascal before buying the IBM was absolutely zero. At the time there were only 2 choices of languages for the PC, BASIC and Pascal. And BASIC was still at the interpretive level, the compiler version had not been released. My mainframe programming had been in PL/1 and I had painfully gone through the transition from unstructured to structured programming. Pascal was obviously the only choice at the time. It is a powerful structured language that lends itself to good readable code. As far as the languages you can buy now for the IBM, the 'C' language is definately a strong competitor. Both Pascal and 'C' are powerful and structured and given a choice between the two I would have to flip a coin. 'C' seems to have 2 advantages. It seems to be coupled to the new wave of UNIX-like systems that are arriving on the micro computer scene. Many larger software houses do much of their development work in 'C' on a bigger computer, then convert it to the appropriate target machine code. The other major advantage of 'C' (or disadvantage) is that it stresses power over safety. It lets you do a lot of things that in a business shop would be unadvisable but in a system programming environment are considered acceptable. Pascal's advantage over 'C' is exactly the reverse. Pascal sacrifices a little of the power for safety. So if you ask me today which language I would choose it would be a toss up.

PJ: What was your approach to learning Pascal?

PETER: My approach to learning Pascal was to just read the IBM manual cover to cover once or twice. The major concepts were clear relatively quickly and I find myself unable to read tutorial books in general. Certainly I missed a lot of details the first time through. I realize that for most people this sort of approach wouldn't be feasible. So I cannot actually recommend any books to go to for assistance in understanding the language. The IBM manual is totally untutorial in its approach.

PJ: What were your major frustrations?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PETER: The compiler did not offer any major stumbling blocks. Like most people however I did encounter a lot of frustration when I made very minor changes to some Pascal code and it would shut down the compiler. It did take me a while to learn the detailed style and flavor of Pascal. I didn't have the experience many programmers have of coming face to face with total mysteries. How the heck did that happen? Is it me or the code or the compiler? This reflects the fact that the Pascal compiler is relatively bug free. A very excellent product.

PJ: How is your system configured?

PETER: My configuration during the development of the Utilities was a very ordinary mid-level machine, nothing exotic on it at all. A IBM monochrome monitor, 128K of memory (64K IBM board), which is the minimum for the compiler, 2 regular disk drives, an Epson printer which I hardly ever used because I usually do my work on the screen. I started out using EDLIN as my editor but quickly changed to VEDIT. It is a very strong program editor and is sufficiently capable as a word-processor too. I have just completed a book for publication and wrote it totally using VEDIT. Once I had a good editor it increased my programming speed tremendously. Now my system is a little more exotic, with a 1/2 meg board from Tall Trees.

PJ: How long did it take for you to feel comfortable with the IBM and Pascal?

PETER: Roughly speaking it took me a month working weekends and evenings to get a handle on the machine as a whole, as well as DOS and Pascal. Then about a month and a half later I realized the need to have a DISKLOOK type program. The actual development of the DISKLOOK program took about a month.

PJ: There was a noticable difference between the first versions of DISKLOOK in terms of speed and size and the later ones.

PETER: The initial version of DISKLOOK was rather large . The screen display, was also quite fast. This was a result of building a complete screen off stage in a separate 4K of memory and then using a repeated move instruction that Pascal gives you access to. I would take this completed block of information and MOVE it to the display screen. The later versions have been reduced substantially in terms of size while maintaining a negligible loss in screen speed. As my Pascal code became more efficient the size was reduced. The 64K boundary also dictated the need to reduce the size. The speed that this provided cost an additional 4K of extra memory space.

PJ: Are the current versions the same?

PETER: In the newer versions the information was built directly into the display memory which reduced the overall memory requirements by 4K with only a barely noticable loss in speed. The user is actually seeing the building of the display in progress. The color graphics adapter has additional onboard memory that allows for extra display page areas which allow the programmer to switch from one to the other. One of the things that I point out in my new book on advance features of the IBM PC is the advantage of these extra pages, but since many programs are designed to run on both monitors this feature cannot often be easily utilized. But as the first versions of DISKLOOK demonstrate, you can just set aside your own 4K of memory and produce the same results. It is an impressive technique to change an information screen so rapidly.

PJ: Are you working on other software?

PETER: Right now I have 2 other projects in the works associated with my new book. There are an awful lot of service routines in ROM. Literally hundreds of interface routines that are normally not accessible to anyone but assembly language programmers. The software I'm writing will make these features available to Pascal or compiled BASIC programmers. Their presense is indicated or hinted at by the better software on the market. But there are still routines that are rarely used at all. For example one of the video services provides a way to scroll separate windows on the screen that are defined by the locations of their opposite corners. The routine allows you to scroll old information out of the window (roll it up and down, etc.) These access routines I'm writing now will be available as object code so Pascal and compiled BASIC users can call them, as well as in assembler source listings to allow a knowledgeable programmer to alter it for their own purposes. There will also be some additional Pascal code to make their integration easier if you are using the language.

PJ: Would you contribute some pieces of code?

PETER: If you would like I could provide a section of one of my programs that would serve as a good model of what structured Pascal is like as well as supplying a useful function that not everyone may know how to do.... perhaps the video I/O routines that give some snappy screen displays. /PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Dear PJ,

Here is the information that I promised you.
First, on batch files:
Like most IBM/PC users, I quickly discovered that using batch files was a great convenience, particularly for a multi-stage operation, like compiling Pascal programs. Here is how I organized mine. There are five stages to working on an IBM Pascal program, and I set up batch files for each-- instead of giving them meaningful names, I just numbered them:

0 invokes the program editor (I happen to use Vedit)
1 runs the PAS1 stage of the compiler (does syntax checking)
2 runs the PAS2 stage of the compiler (completes the compilation)
3 does the link-editing
4 test-runs the completed program

The advantage of these terse, one digit names, is that they can be combined easily. I have separate batch files for each of the five, and also quite a few combination files (012 to do a complete edit and compile, 12 to do compiling only, and so forth). I have found that at various times different combinations of the five stages are useful, so I keep a complete set of many different combinations. Which ones you might want to use depend on your working style and the capacity of your diskettes. The two most useful of the combined batch files are 01, used to edit and syntax-check programs, and 23, used to finish the compilation and link-edit a ready-to-run program. The exact batch files that I use have a lot of peculiarities matched to my own needs. Here is a revised version that ought to be useful for many people who use Pascal on the IBM/PC:
Batch file "0.BAT" to run the VEDIT editor:

```
REM                     EDIT THE PASCAL PROGRAM %1
VEDIT B:%1.PAS
REM                     JUST EDITED THE PASCAL PROGRAM %1
```

Batch file "1.BAT" to do a first-stage Pascal compile:

```
REM                     COMPILE THE PASCAL PROGRAM %1
B:
A:PAS1 %1.PAS,%1.OBJ,NUL,NUL
A:
REM                     JUST COMPILED THE PASCAL PROGRAM %1
```

Batch file "2.BAT" to do a second-stage Pascal compile:

```
B:
A:PAS2
A:
```

Batch file "3.BAT" to link edit the program

```
REM                     LINK-EDIT THE PASCAL PROGRAM %1
LINK  B:%1.OBJ,B:%1.EXE,CON;
REM                     JUST LINK-EDITED THE PASCAL PROGRAM %1
```

Batch file "4.BAT" to run the new program

```
REM                     TEST THE PASCAL PROGRAM %1
B:%1
REM                     JUST TESTED THE PASCAL PROGRAM %1
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here is a sample of some Pascal code that ought to be useful in two ways: it illustrates reasonably good structured programming style, and it shows how to do direct output the the screen of the IBM/PC:

```
{   Pascal routines for direct screen output                          }
{   (C) Copyright 1982, Peter Norton                                  }
{   These routines illustrate how to do direct output to the screen   }
{   You can adapt them to your own particular preferred style         }
{   Here is an example of how to use them:                            }

{       set_video_address;                                            }
{       clear_screen;                                                 }
{       row := 5; column := 5; output_string := 'Hello';              }
{       display_output_string;                                        }
{       display_this ('Error!',10,25,alert);                          }

module Screen_output_IBM_PC;

{   These service routines contain most or all of what you will need  }
{   to do direct output to the IBM/PC display screens.                }
{=====================================================================}

{   Let's start by defining the types we need, beginning with the     }
{   screen:                                                           }

type

 screen_position_pair_type = (character_byte, attribute_byte);

 normal_screen_type = array [1..25,1..80,character_byte..attribute_byte]
                      of char;

   { Notes-- our screen is set up to number positions from 1; if you  }
   {   prefer counting from zero, then change it accordingly.         }
   {   For the less-common 40 column display, use this:               }

narrow_screen_type =  array [1..25,1..40,character_byte..attribute_byte]
                      of char;

   ls255 = lstring (255);

{=====================================================================}

{   Next, here are the variables that we will be using:               }

var [static]

{   To get access to the screen, we need an address variable:         }
   screen_pointer     : ads of normal_screen_type;

   current_attribute : char;           {used to specify the display attr.}
   row, column       : integer;        {used to specify display location }
   output_string     : ls255;          {used to hold data to be displayed}

   page              : word;           {used to specify which display page}
   eighty_col_mode   : boolean;        {used to specify 80/40 column mode}
   monochrome_display: boolean;        {used to specify monochrome or color}

{=====================================================================}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
{ Next, we'll define some of the more common attributes used;          }
{ You should add any that you want to use.                             }

const                                  { Monochrome   /  Color-graphics  }

  normal_attrib    =  chr   (7);    { normal        /  white on black  }
  reverse_video    =  chr (112);    { reverse       /  black on white  }
  bright           =  chr  (15);    { bright        /  bright, maybe   }
  blinking         =  chr (135);    { blinking      /  blinking w-o-b  }
  alert            =  chr (140);    { bright-blink  /  blinking red    }

{=======================================================================}
{  First things first-- we must set our address pointer to the         }
{  right screen buffer, for monochrome or color-graphics.              }
{  Your programs must either find this out (from the user, or the      }
{  operating system) or assume one or the other.                       }
{  A simple assembly-language routine can do this.                     }

procedure set_video_address;
  begin
    if monochrome_display then
      begin
        screen_pointer.s := #B000; {segment paragraph for monochrome}
        screen_pointer.r := 0;
      end
    else
      begin
        screen_pointer.s := #B800; {segment paragraph for color-graphics}
        screen_pointer.r := 0;
      end;
  end;

{=======================================================================}
{  The next routine clears the screen-- with or without an  assembly   }
{  language routine.                                                    }

procedure clear_screen;
  begin
  { if you have the assembly language routines, just to this: }
  { clear_reset;  }
    return;
  {  otherwise, we'll clear the screen the hard way :          }
    for row := 1 to 25 do
      for column := 1 to 80 do
        begin
          screen_pointer ^ [row,column,character_byte] := ' ';
          screen_pointer ^ [row,column,attribute_byte] := normal_attrib;
        end;
  end;

{=======================================================================}
{ The next routine is the work-horse of display-- it puts data on      }
{ the screen, as needed.  This routine picks up its parameters of      }
{ where to put things on the screen, what to put, and which attri-     }
{ bute to use, from variables; if you wish, you could change that      }
{ to have the information actually passed as parameters, but you       }
{ will probably find the method shown to be better.                    }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
procedure display_output_string;
  var [static]
    i : integer;

  begin
    for i := 1 to ord(output_string.len) do
      begin
        screen_pointer ^ [row,column,character_byte] := output_string [i];
        screen_pointer ^ [row,column,attribute_byte] := current_attribute;
      end;
  end;

{=============================================================================}
{ The next routine integrates the setting of the row, column,                 }
{ attribute, and output string into a single procedure call.                  }
{ You may prefer setting these items independently, or specifying             }
{ them in each output call.                                                   }

procedure display_this (const this : ls255; parm_row, parm_col : integer;
                        parm_attrib : char);

  begin
    row := parm_row;
    column := parm_col;
    current_attribute := parm_attrib;
    output_string := this;
    display_output_string;
  end;

{=============================================================================}
{ If you wish to use the color-graphics multi-page feature, this              }
{ procedure will set the screen_pointer to the appropriate paragraph          }

procedure set_pointer_for_page;
  begin
    if eighty_col_mode then
      screen_pointer.s := #B800 + page * 4096
    else
      screen_pointer.s := #B800 + page * 2048
  end;


end.
```

Sincerely,

Peter

Peter Norton

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IBM Personal Computer COBOL

by Rich Rohde

Since the beginning of September I have been using COBOL for the IBM Personal Computer to develop software for business applications. This version of COBOL conforms to the 'Low-Intermediate' level of the American National Standard X3.23-1974. Some of the features of this version of COBOL are:

1. Processing and the management of sequential, relative and indexed I/O (ISAM) files. The LINAGE clause for print files is also provided. Although not directly stated some form of communication is provided by means of specifying COM1 in the VALUE OF FILE-ID clause in the file description.

2. Interprogram communications are handled in two methods, the first being the use of the CALL statement passing parameters to the called program, where the called program must have a LINKAGE SECTION and a USING phrase on the PROCEDURE DIVISION statement, the program must also include an EXIT PROGRAM statement as opposed to a STOP RUN, to return to the calling program. The second method is to use the CHAIN statement where the called program is a link edited executable program. When the CHAIN statement is used, the complete file name must be specified, including the file extension.

3. SORT/MERGE is NOT supported.

4. REPORT WRITER is NOT supported.

5. Segmentation is supported so that you may write larger programs that can be divided into overlays. Before you may call an overlay (independent segments 50 to 99) all files must be closed and all I/O must be performed in the segment. In an attempt to use segmentation I found that the DISPLAY screen name and ACCEPT screen name did not work in the segment. To get around this undocumented situation, I used the DISPLAY and ACCEPT statement prior to the performing of the segment. It has been brought to my attention the DISPLAY and ACCEPT statements should function in segments and the problem has been reported to IBM as early as June of this year.

6. The feature that I found most attractive was the SCREEN SECTION. Within this section you are able to totally define an output screen with row, column, underline, reverse video and high intensity. With a graphics display colors maybe used. The definition of data to be displayed from another data name is supported. To have a defined display appear on you screen, simply use the DISPLAY statement. Data names may also be included to tell COBOL where data being accepted from the screen are to go. Programs written in IBM Personal Computer COBOL require the COBRUN.EXE runtime module. If you are writing applications to be distributed, a license agreement with IBM will be needed. I sent for my agreement in September and received a letter from IBM indicating they are currently working on the final contract terms and distribution procedures offering significant flexiblity and benefits for program developers and users. As of the end of November I have yet to receive any such contract or agreement.

In the FILE-CONTROL you may specify in the SELECT clause a FILE STATUS data name phrase. The data name must be defined in the WORKING-STORAGE SECTION as a PIC X(2). I had defined the file status as a PIC 9(2) and received the compile error message 'UNRECOGNIZABLE ELEMENT IS IGNORED' with the line number of the error being that of the PROCEDURE DIVISION.

There is one reported BUG that I am aware of in the COBOL compiler or runtime modules. This has to do with the order in which the data is moved to a field in a record within the DECLARATIVE SECTION. The following example will help illustrate the problem.

```
.
.
FILE-CONTROL.
    SELECT PARAM-FILE ASSIGN TO DISK
    FILE STATUS IS STATUS-FLAG
    ACCESS MODE IS DYNAMIC
    ORGANIZATION IS RELATIVE
    RELATIVE KEY IS PARAM-RECORD-KEY.
.
.
.
FD  PARAM-FILE
    VALUE OF FILE-ID 'PARAM.DAT'
    LABEL RECORDS ARE STANDARD.
01  PARAM-RECORD.
    05  FIELD-1     PIC X.
    05  FIELD-2     PIC 9(4) COMP-3.
.
.
.
WORKING-STORAGE SECTION.
01  STATUS-FLAG          PIC X(2).
01  PARAM-RECORD-KEY     PIC 9.
.
.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
PROCEDURE DIVISION.
DECLARATIVES.
PARAM-FILE-ERROR SECTION.
    USE AFTER EXCEPTION ON PARAM-FILE.
DISPLAY-STATUS-FLAG.
    IF STATUS-FLAG = '30'
        CLOSE PARAM-FILE
        OPEN OUTPUT PARAM-FILE
        MOVE SPACES TO FIELD-1
        MOVE ZEROS TO FIEDL-2
        MOVE 1 TO PARAM-RECORD-KEY
        WRITE PARAM-RECORD
        CLOSE PARAM-FILE
        OPEN INPUT PARAM-FILE.
END DECLARATIVES.
MAIN-SECTION SECTION.
MAIN-PROCEDURE.
    OPEN I-O PARAM-FILE.
    MOVE 1 TO PARAM-RECORD-KEY.
    READ PARAM-FILE RECORD.
```

In the example above if the file PARAM.DAT does not exist the error will occur when attempting to open the file as I-O and the DECLARATIVES will be performed. As you can see the file is closed then reopened as OUTPUT, the data record initialize and the record number one written. Control is returned to the main line with the file opened as I-O and the record is read, but an error is encountered with the STATUS-FLAG containing a '23' (no record found). After checking the actual contents of record number one by using DEBUG and finding that FIELD-1 was null, I reversed the order in which the record was initialized to the following:

MOVE ZEROS TO FIELD-2
MOVE SPACES TO FIELD-1

then reran the program with complete success.

Development of software using COBOL on a 64K machine is impractical since the primary reason for using COBOL is to write business oriented applications, therefore, I would recommend development on a system with 128K or more. Running applications on a 64K machine must also be carefully considered, since the largest LINKED COBOL program cannot be greater than 28K (COBOL runtime program uses about 32K and DOS uses about 4K). The application programmer must also consider the number of disks to run an application. The disk containing the COBOL runtime program and libraries uses the better part of a 160K disk. I have found when one program chains to another the runtime programs are not reloaded, therefore, once the first program is loaded into memory the runtime disk can be removed. If the system you are developing has any magnitude at all, you should consider 320K disks or even a hard disk where you will have 1 or more megabytes of storage.

In the development of software I have found that my development disk has turned into many disks. After I have compiled a program the listing (.LST) is moved to a LISTING disk, the object program (.OBJ) is moved to a LINKEDIT disk where the program is linked. The linking process produces an executable program (.EXE) and a link map file (.MAP). I then move the EXE files to an EXECUTABLE program disk and the MAP files to a MAP disk. The point I am making is that you must consider the number of disks you will need in the development of a system. Of course the number of disks required is relative to the size of the application you are developing, the smaller the system the fewer the disks, the larger the system the greater number of disks.

If you are an experienced COBOL programmer I would strongly suggest that you consider writing any business applications in the IBM Personal Computer COBOL. The compiler is somewhat time consuming and the management of the source, listings , etc. is distracting, but you will be pleased with the final results of your efforts.

The manual that accompanies the IBM Personal Computer COBOL is easy to follow, although the screen section could be greatly improved. If you have written COBOL programs on other systems, you will find using this version very easy. The manual is divided into ten (10) chapters; 1) Introduction; 2) How to write a COBOL program; 3) Developing a program; 4) Identification division; 5) Environment division; 6) Data division; 7) Procedure division; 8) Data input and output; 9) Table handling by the indexing method and 10) Interprogram communication. Appendix A includes the compile time errors which are quite good, although I have run into some errors which were not included. Also included are the runtime errors which are very complete, besides having the line number which caused the runtime error (NOTE: the line number appearing can be suppressed by a compile time option allowing you to reduce the size of your program).

This version of COBOL is as powerful as the COBOL on larger systems and close to being 100% compatable. I would strongly recommend the purchase (which is currently $700 at COMPUTERLAND) for business applications. □

RICH ROHDE has been in data processing for 11 years in both the public and private sectors. He has written programs in most languages for user applications and internal computer functions. In the past 5 years, he has been using COBOL for writing data base and information management systems to allow application programmers to develop software more effectively. Currently he is running his own software development company writing applications in COBOL on the IBM Personal Computer.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ FORUM

by Frankie Detesto

Bench Marks & Wart Hogs: Bench mark testing seems to be more and more in vogue, and many people I have talked to recently have argued that "my language is better than your language" based upon them. Just for the sake of fueling the fire in this area, let's just make up a bench mark and form some conclusions.

How does this look:

Assembly	1 second	(the base unit)
FORTH	4 seconds	
"C"	5 seconds	
Pascal	7 seconds	
BASIC (compiled)	10 seconds	
FORTRAN	15 seconds	
COBOL	17 seconds	
BASIC (inter)	38 seconds	

The above times coupled with the philosophy that some languages ARE better than others could give us a lot of room toshootthe bull so to speak. Of course you know that all of the tests were run with exactly the same piece of code translated into the appropriate language (?).

So what did that bench mark do for you? Are you going to go right out and buy a "C" compiler? Are you going to trade in your BASIC compiler for a Pascal compiler because it is faster? Maybe. This sort of bench mark has a definate curiousity value, but when you tell me that the pieces of code are the same, or that the statements executed are pretty much identical then I think I'm going to get off the boat. I do not hold to the philosophy that one language is "better" than another. If I want to draw some graphic circles I will gladly choose BASIC. If absolute speed is essential, even above development time then assembly would be the best choice, and if teaching someone (forcing someone) to write structured program design was tanamount then Pascal would be the one. But I'll let you in on a little secret. I do not know all of those languages like the back of my hand. It is a long laborious process to COMPLETELY learn a programming language. If it were a piece of cake we would be experts in all of them.

Every language is a trade off. In assembly, speed and access to the hardware are at the top of the list. In Pascal structure and modular design combined with a reasonable amount of hardware access give you a very balanced tool. You are still stuck with a compiler (oops the P-system people are knocking at my door.) Some languages allow you to be sloppy and others force you to be neat and exact. But the "oranges and bananas" approach to bench marks only fuels the "my language is better than your language" fist fights which I'm sure you have all heard. Spear-headed more often than not by the Pascal converts.

Yes, I do think bench marks have their place. Here would be a more appropriate comparision:

IBM Pascal	10 seconds
Pascal MT +	9 seconds
JRT Pascal	12 seconds
P-system	14 seconds

The actual times are fictitious, but they are all Pascals and the code is letter for letter the same. If I know Pascal and want to pick out one with speed and the test code dealt with a research problem I would doing a lot of then there is a very definate reason to choose one over the other. This is still a thorny issue. I would hope that 20 or 30 pieces of code were compared to give an overall view of each language. Numeric and string handling capabilities vary greatly from compiler to compiler. Cut down on the precision and gain some speed.

One of my favorite lines from a bench mark article goes like this, "All things being equal I would choose. . . " . I am glad to know that someone is fluent in 10 programming languages. Yep, if I had the choice between a dead horse and a dead cow I would choose the dead horse. All things being equal of course. Perhaps the lesson to be learned is the "top down" approach to programming. Start at the top of the bench mark list and work down. First learn assembly then FORTH then "C" etc. That way no one can ever say their language is faster than yours. If that isn't top down programming then I don't know what is.

FUNCTION KEYS: The special function keys on the IBM PC have their place, and also their abuse. Not too long ago I felt they were a necessity and every computer should have 10 or 20 of the things. Experience tempers all great ideas. If you don't agree just talk to a touch typist about making them leave home position to find one of the luxury keys. Word Star was not too much fun to learn. The command structure is heavily control this, control that. My dissatisfaction was loud and consistant everytime I was asked about the word-processor. Then I met a typist who was fluent with the control sequences and changed my mind. Hunt and peckers love function keys alot, whereas the good typist view them as more of a distraction. Function keys should be used for complete functions or menu options. Help screens and special command series fit well too. But if a word-processor is trying to please all of us, then both options should be available. Have a control series for the fast typist and duplicate many of the same commands with the function keys.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOOLS OF THE TRADE

by Greg Estes

Programming in any language goes through a process of evolution. The standard tools of BASIC we will cover in this column are very useful in themselves, but the real concern is establishing the awareness how our software changes and improves with age, and attention, and, most of all, contact with other good software writers.

Our first focus is often just "getting the job done," no matter what the cost in code size or structure. Then as our skills and understanding of the language improve we are able to reduce the size of the program while still keeping or increasing the clarity of its function. We will quickly cover the evolution of some single line BASIC tools before moving on to larger ones like input routines.

When a YES or NO question is asked a user we must determine which response was made and execute the appropriate choice. Initially we code it like this:

```
100 INPUT "Do you want to try again (Y/N)";A$
110 IF A$="y" THEN GOSUB 500
120 IF A$="Y" THEN GOSUB 500
130 IF A$="n" THEN GOSUB 600
140 IF A$="N" THEN GOSUB 600
150 GOTO 100
```

Lots of code. Soon we figure out that we can shorten it slightly by checking each answer in one line:

```
110 IF A$="y" OR A$="y" THE GOSUB 500
120 IF A$="n" OR A$="N" THE GOSUB 600
130 GOTO 100
```

This is better, and for a yes or no question certainly suffices. But what we if we had a menu with 5 choices and the options were A, B, C, D, or E ?

Microsoft gave us a standard tool for this, the INSTR function. It searches a string for a character and returns the number of where it found it, or a zero if it failed. Our yes/no question can now be coded like this:

```
110 NUMBER=INSTR(1,"yYnN",A$)
120 ON NUMBER GOSUB 500,500,600,600
130 GOTO 100
```

As time goes on we notice that the INSTR function itself gives us the number we need, so why bother with the NUMBER variable at all. This enables us to handle a list of menu options (say A-E) in only one line:

```
110 ON INSTR(1,"aAbBcCdDeE",A$) gosub 500,500,600,600,700,-
        700,800,800,900,900,
120 GOTO 100
```

Every function in BASIC has numerous uses. As our experience and creativity in a language evolve we discover new uses for old functions. Many such ideas are passed from programmer to programmer with improvements and new tricks added along the way.

UPPERCASE AND LOWERCASE: There are many times we want to convert pieces of data to uppercase from lowercase. (In a mailing list program the 2 letter state field should always be forced to uppercase even if the user forgets, often the first letter of first or last names are also forced to uppercase, and especially when building index files or searching for a matching name you normally want to compare fields both of which are uppercased to catch every occurance. "Johnson" should match "JOHNSON" unless otherwise indicated.) In order to do this we check to see if a letter is within the lowercase a-z range, and if it is just make it uppercase by subtracting 32 from its ASC value. Look at appendix G in the BASIC manual. We have several functions we can put together to do this. The ASC and CHR$ functions. For example:

```
200 C$="b"
210 C$=CHR$(ASC(C$)-32)
```

The result being a 'b' is changed to 'B'. With larger variables we need to do it in a loop. The LEN function gives us the number of characters in our variable. And we are going to put the letters one at a time into a C$ field to check them. The MID$ function which allows us to extract pieces of words. Let's assume we have a state code called ST$ and we want to convert the whole thing to uppercase.

```
200 FOR I = 1 TO LEN(ST$)          (for length of state)
210 C$=MID$(ST$,I,1)               (get it)
220 IF C$="" THEN GOTO 250         (possible error)
230 IF C$)="a" AND C$(="z" THEN C$=CHR$(ASC(C$)-32)
240 MID$(ST$,I,1)=C$               (put it back)
250 NEXT I
```

IBM PC BASIC from Microsoft ALMOST gave us a complete tool to do this with the Define Function capability. The were a few grumblings from the programming community when the DEF FN (defining functions) was only allowed a one equation length. The problem is apparent when we want to force a field (our STATE field) to uppercase. It would be nice to check and see if a letter was already capitalized, and then capitaltize it all in one fell swoop. This would require 2 statements, and the DEF FN only allows us 1. So we do the best we can. Defining a function we get:

```
20 DEF FNCAP$(A$)=CHR$(ASC(C$)-32)
```

This method is ideal for numeric equations but can often be useful with strings. Remember to end the definition with a $ if it is a string definition and put the code physically ahead of the function. (NOTE: I spent 2 hours one Saturday morning wondering why one program line kept getting a SYNTAX error. I was writing a mailing list program and I had changed a variable name from FIRSTNAME$ to FNAME$. The only variable name you cannot use other than reserved words is a variable beginning with FN because Microsoft BASIC considers it to be a function that has been previously defined. So do not ever use a variable that begins with FN unless it it IS a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

special user defined function.) Line 230 of the above statement could then be rewritten using the defined function.

```
230 IF C$)="a" AND C$(="z" THEN C$=FNCAP$(C$)
```

DIVIDING BY ZERO: This is a perpetual problem. We should always check any number to be used for division to see if it is equal to zero.

```
200 IF X=0 THE GOTO 220
210 AMT=AMT/X
220 .........
```

You can abbreviate this by just saying "IF X" which is the equivalent of saying "IF X = 0".The "IF" here is going to see if the variable or equation is TRUE or FALSE. TRUE means a non-zero and FALSE means a zero value. Rewriting we can say:

```
200 IF X THEN AMT=AMT/X    ELSE  AMT=0 ..... and so forth
```

MONTH/DAY/YEAR: Dealing with dates is as much fun as giving your cat a bath. One approach is to convert dates from our MM/DD/YY calendar (Gregorian) to a Jullian date or absolute number. A chunk of code that takes both time and space. Since a lot of information our computer handles is dated, and it is something we have to SORT on, this situation is repeatedly staring us in the face. A simpler approach that solves much of the dilemma is to move the Year to the front of the date so you have a YEAR/MONTH/DAY field. So whether the field is saved as a string or a number it still sorts out correctly by date (Don't forget the zeros.)

```
12/01/82 is stored as "821201"
01/02/83 is stored as "830102"   etc.
```

HOUSEKEEPING: Often you can spot a good piece of code written by an experienced programmer by looking at the level of his housekeeping or initialization at the beginning of the program. We will mention a few and explain their usage.

```
20 DEFINT A-Z : TRUE=-1 : FALSE=0 :CLS :SCREEN 0,0,0
30 ESC$=CHR$(27) : RTN$=CHR$(13) : BS$=CHR$(8)
40 CTLB$=CHR$(3) : FOR I = 1 TO 10 : KEY I,"" : NEXT I
```

The DEF INT makes all the numeric variables you use of type integer unless otherwise stated. This saves a lot of space should you use many numbers and also speeds things up. The next steps are to set up a few variables to make our program more readable. You can set a variable to TRUE or FALSE at one place in the program and then check it later.

```
100 OVERDUE=TRUE
110 NEWSTUDENT=FALSE
.
.
.
2005 IF OVERDUE=TRUE THEN .............
2510 IF NEWSTUDENT=FALSE THEN ......
(or you can say)
2505 IF OVERDUE THEN ..........
2510 IF NOT NEWSTUDENT THEN ...
```

This really helps when you are trying to narrow down program bugs and find out what you are doing. Along with this we are going to regularly check the user input to see if they typed a Backspace (BS$) or a Return (RTN$) or a Control Break (CTLB$). We also set the function keys to nothing so they won't cause us trouble.

INPUT ROUTINE: We are going to spend the remainder of this column examining the evolution of an input routine, our next issue will cover it in full. An input routine is a MAJOR building block of decent software. It is the one foundation upon which decent micro computer software is built. A user spends most of their time answering questions and inputing data based upon your input routine. A poor input routine will make an otherwise excellant piece of software unusable. It is a routine that will be in all your programs involving input, whether numeric or string. We will begin at the beginning.

```
100 INPUT "How old are you ?";A
```

This works . . . sort of. But we want to produce some quality programs. The drawbacks of this input technique are well known. The users can type 100 numbers if they want to, or type in a letter which causes a 'Redo from start' and destroys our screen. We have very little control over the cursor.

```
100 INPUT "How old are you";A$
110 A=VAL(A$)
120 IF A(1 OR A)100 THEN GOTO 100
```

One small step for mankind here. At least we got rid of the "Redo from start" message. The A = VAL(A$) produces the number again from the string. Also we looped around if the number was not within the range. Still the user can hit the ESCAPE key and erase the screen line. Not good at all. What we would like is a statement that traps every piece of input and allows us to keep or throw away what we want. That function is INKEY$. This little gem gives us every keystroke from the keyboard one at a time. The computer is in fact perpetually reading the keyboard. (If you doubt this then try using INKEY$ and set the TRON key . . . F7) All we have to do is build a first class routine around it and we are well on our way to producing 'friendly' and easy to use software.

Let's improve our input by locating the screen line and then using INKEY$. We must accept the keystroke from INKEY$ to a string variable, let's call it INK$.

```
50 LOCATE 10,10: PRINT "Do you want to try again ? (Y/N)";
60 GOSUB 1000
70 END

1000 REM ———— INPUT ROUTINE
1010 INK$=INKEY$ : IF INK$="" THEN GOTO 1010
1020 IF INK$=.....sort out what you want, and return if done
.
.
.
1200 GOTO 1010 ..... if more than 1 char desired, loop
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Line 1010 sets up the endless loop until a key is hit. So once we have something, what will we do with it? We have trapped everthing from the ESC key to the Control Break key to the Backspace key. The decisions you make here determine how friendly USERS consider your software. So we will throw out a few keys and keep some others to make a minimal routine that is understandable before considering a full length version next time. It all boils down to one concept: keep collecting the keystrokes from the user until they hit the return key , that means they are all done. If a keystroke is good, save it, if it is not, ignore it. Fleshing out our routine a little we will add a WORD$ variable. This is the place where the keystrokes are saved.

```
10 REM-------HOUSEKEEPING
20 DEFINT A-Z : TRUE=-1 : FALSE=0 :CLS
30 ESC$=CHR$(27) : RTN$=CHR$(13) : BS$=CHR$(8)
40 CTLB$=CHR$(3)
50'----------USER INPUT
60 LOCATE 10,10: PRINT "What is your Name    ?   ";
70 GOSUB 1000
80 PRINT WORD$
90 END
1000 REM -------INPUT ROUTINE
1010 INK$=INKEY$ : IF INK$="" THEN GOTO 1010
1020 IF INK$=RTN$ THEN RETURN
1090 WORD$=WORD$+INK$
1200 GOTO 1010  ........ if more than 1 char desired, loop
```

Type this short program in and watch what happens. When you hit keys nothing shows up. INKEY$ sends nothing to the screen. This is good for password protection and the like but not for user input, they need to see what they are doing. Once you hit the RETURN key the subroutine ends and you do eventually print the WORD$. Here we begin adding, line by line, what we need to the skillfully flesh out our input routine. Let's add some printing.(Don't forget the semi-colon.)

```
1080 PRINT INK$;
```

Seems to work. The ESC and Backspace keys look funny. We will stop them later. But how about the Control Break key? We have stopped the break exit unless the user is really fast. How about one more addition before we close shop for the day. Make the user stop at 10 characters. If the 10th letter is NOT a return we will beep and try again.

```
1030 IF LEN(WORD$)=10 THEN BEEP :GOTO 1010
```

(Delete line 80 too.)
All together it looks like this:

```
1000 REM -------INPUT ROUTINE
1010 INK$=INKEY$ : IF INK$="" THEN GOTO 1010
1020 IF INK$=RTN$ THEN RETURN
1030 IF LEN(WORD$)=10 THEN BEEP :GOTO 1010
1080 PRINT INK$;
1090 WORD$=WORD$+INK$
1200 GOTO 1010  ........ if more than 1 char desired, loop
```

This is only the start of a good input routine. We should ignore other keys and print a prompt line and allow the user to backspace and correct a mistake. Using the MID$ function we could add and subtract characters from our WORD$. Next issue we will put a complete routine together with all the bells and whistles that you can plug into any program and have first class, bug free input. Stay tuned. . . □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C

There is a great deal of interest in the C language. It is lurking in the background of many major software firms (Microsoft, Bell Systems Lab, Tektronix, etc.) Certain big name software marketing firms prefer applications in C because it is 16 bit related and transportable (Lifeboat for example.) The language is very much allied with UNIX since UNIX is written in C. There is only one definative book on C written by Kernighan and Ritchie called The C Programming Language. But after we have purchased and read that book, and learned the basics of C, where do we go from there?

The big SW houses have large development systems, mainframes with UNIX or UNIX like environments which provide an ideal setting for programming. Their software authors write their applications in C and use the appropriate compiler to translate them into the target machine code. The idea of a room full of programmers working on 10 different computers is a thing of the past. Also their is the ultimate reality of writing your applications in the same language as your operating system. And behind it all is the fact that UNIX and C were developed, not with the idea of being "user friendly" but with the concept of being "programmer friendly," if that is possible.

C is a relatively low-level language dealing with machine words like most assembly languages do. This makes it suitable for writing operating systems. However it will also handle characters, integers and floating-point numbers. This makes it possible to supply libraries in C to handle the string manipulation of higher-level languages without sacrificing the low-level capabilities. Library routines must handle I/O which is one criticism of the language. Pascal resorts to the same "library" approach supplying much need functions, screen cursor control being only one of them. Higher-level languages spare us the trouble of writing and dealing with these accessory libraries, while at the same time making our code more machine dependent. There is always a trade off.

The structure of C is very much like Pascal but it is more syntax ridden. Example:

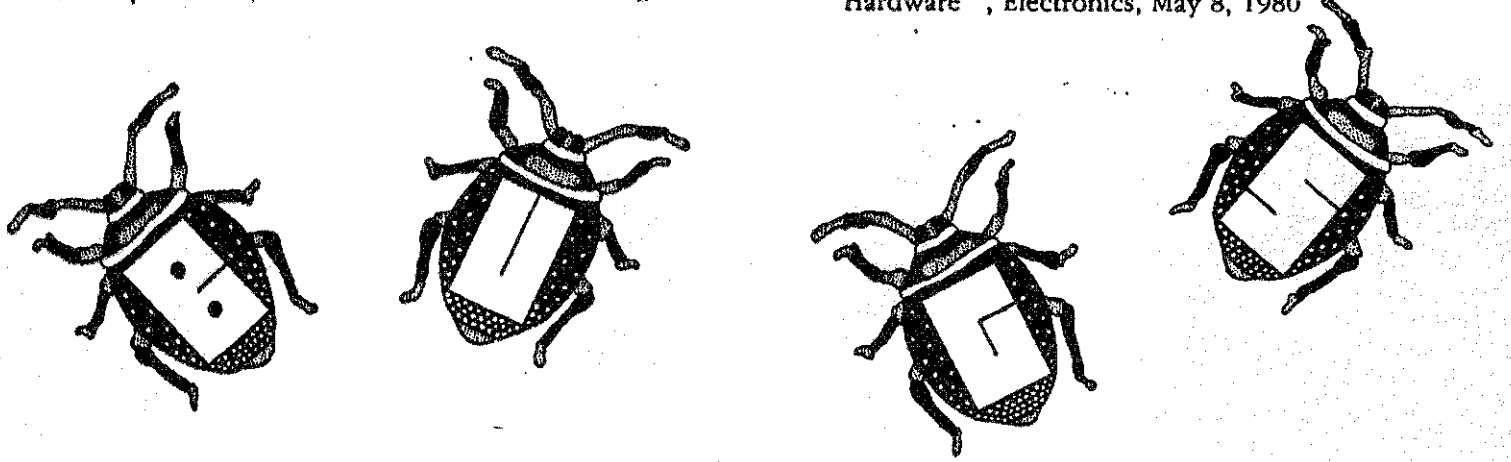
*while ((*d + + = *s + +)! = '/0').*

C contains the usual control flow statements (if-then-else, while,for, and case) like most modern languages. Many data types are supplied as well as a definition facility to create new types.

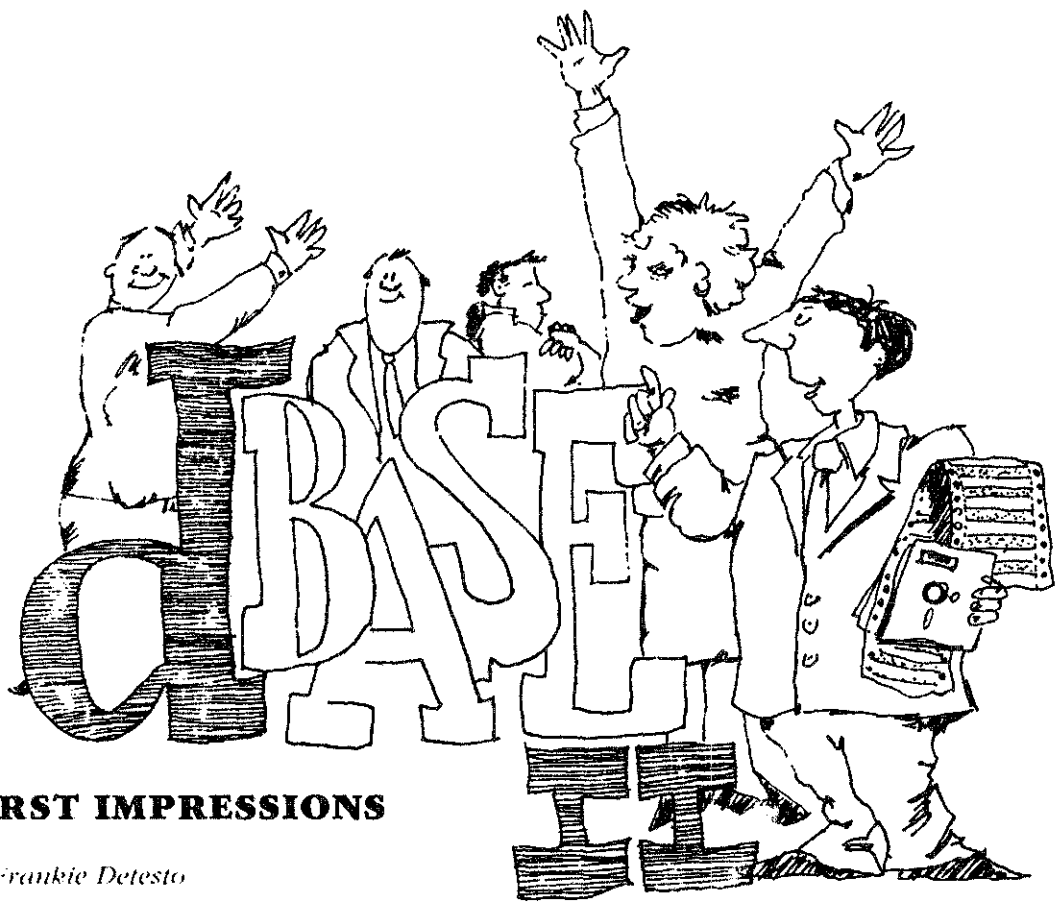
Our approach to covering C will be to combine a general knowledge of learning more about this powerful language while directly considering it's possiblities on the IBM PC. Is it a viable language for general purpose applications or just operating systems or both? There is a very inexpensive C compiler called smallC that we will be reviewing as well as discussing guidelines for picking out a full C compiler. The implementations of C on the IBM vary. Some only parially cover the full C standard as described in Kernighan and Ritchie's book. A major difference being whether floating-point representation is included. But then that is often not needed in normal applications. /PJ

REFERENCES

1. The Bell System Technical Journal, Volume 57 July-August 1978, Number 6, Part 2 (For current subscription rates write The Bell System Technical Journal, Bell Laboratories, 600 Mountain Ave., Murray Hill, N.J. 07974) .
2. Dr. Dobbs Journal, Number 45, May 1980, Volume 5, Issue 5, The C Programming Language (P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94025)
3. Brain Kernighan/Dennis Ritchie,The C Programming Language,Prentic-Hall, Englewood Cliffs, New Jersey, 07632
4. Krieger and Plauger, "C Language s Grip on Hardware" , Electronics, May 8, 1980



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



FIRST IMPRESSIONS

by Frankie Detesto

If you have NEVER tried using a Data Base Manager like dBASEII then this first paragraph is just for you. I consider myself to be your normal sceptical, not easily impressed, don't tell me show me kind of person. Writing software packages for a living tends to give you a facility for producing programs quickly, and after purchasing the IBM PC I decided I needed a few programs to handle the workload on the home front. These included your standard mailing list, check balancing, record keeping type of programs. But indecision kept creeping in about whether 'to write or not to write' the necessary software. Most of the packages are available for less than $100, and if I limited myself to 3 hours I definately could not come up with a mailing list program with flexibility and sorted output. So I broke down and bought a mailing list program. Then the dilemma returned. The mailing list program was not up to the calibre I was used to, and spending more money to try and find one that was left me lukewarm. Only 2 options left. Write the programs myself or really take a chance and fork over the bucks for a Data Base Manager (dBASEII.) I flipped a coin and dBASEII won. So $4?? dollars lighter (see Infoworld) and 3 days later I had a copy of dBASEII. What were my first impressions? Well within 1 hour I had skimmed the enormous manual and began altering their sample mailing list programs to fit the fields I needed. By the end of the 2nd hour, after typing in 20 addresses or so I began playing around with the interpretive mode with commands like:

```
DISPLAY ALL
BROWSE
COUNT FOR ZIP = "98"
REPLACE NAME WITH "BUDDY EMMONS"
```

And frankly I haven't been so excited since we landed on the moon. To say that a normal programmer can write standard programs in 1/10 of the time is a GROSS understatement. I wanted to walk down the street in our neighborhood, knock on doors, and tell everybody about dBASEII. The bottom line is this, you should get a copy of dBASEII if you fall within either of the following categories:

1. You can program and need to crank out a of lot programs for yourself, your friends, or small businesses.

2. You are new to programming and have come to the grim realization that it will take you a year of weekends to ever complete a good BASIC program and you could sure use several working programs right now.

Of all the software I have ever purchased, dBASEII, VISICALC, and VEDIT are at the top of the list. The rest of this column is for those who know what I mean.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Ashton-Tate should have included a one page 'Do this first' sheet with the documentation. They did tell you to back up the programs and then left you sitting there with a sample disk and an enormous manual. They should have told you something along the lines of "Insert the diskette and type DBASE MAIL and play around a little while." After entering 10 or 20 addresses, exit to the dBASE prompt and experiment with the commands. The cursor control table is on page 19.

DISPLAY
LIST
COUNT
BROWSE

Once you have had some fun, exit to DOS and print the mail files ending in .PRG (these are the program files.) They will show you how incredibly small your programs can be and still get the job done. You should mentally replace .CMD with .PRG when you are reading the manual. Ashton-Tate's search and replace function must not work with the word-processor that contains their documentation. No doubt they were rushing as fast as they could to convert dBASEII to the IBM PC.

After reading the red section of the manual you will be pretty familiar with a lot of the "show and tell" statements. The command that should have been explained first and took me hours to hunt up was:

DEFAULT TO B:

Maybe I missed it. This allows you to access all your files on the B: drive. Improvements to the screen and edit work in the sample program can be improved with these:

SET CONFIRM ON (you must hit return after each field)
SET BELL OFF (no bell thank you)
SET COLON OFF (personal preference)
.... OFF .... (In a command line supresses the REC #)

There is an excellent book out that serves as a tutorial for dBASEII from SoftwareBlanc. It costs $29 dollars and is well worth it. If you are 10 year veteran of DP shops then it may not help you much, but if you want a well structured programming guide from the bottom up then it more than fills the bill. SoftwareBlanc also gives seminars on dBASEII programming throughout the US.

A few more handy commands, (4 letters are enough)

DISP FILES LIKE "*.BAS"
REPLACE STATE WITH !(STATE) (the ! makes it uppercase)
ERASE (clear the screen)
SET PRINT ON (print the screen info)
SET TALK OFF (the SKIP command displays REC # so set TALK off if you don't want REC #'s.)

The BROWSE editing features seem to have some bugs. I had trouble staying in field number one and I inadvertantly ended up back in DOS. Hopefully Ashton-Tate will use the standard IBM PC keys for their future updates so we are not handcuffed to the Ctrl this and that memory game. Some of the combinations seem a little confusing when you go from EDIT to BROWSE to MODIFY modes.

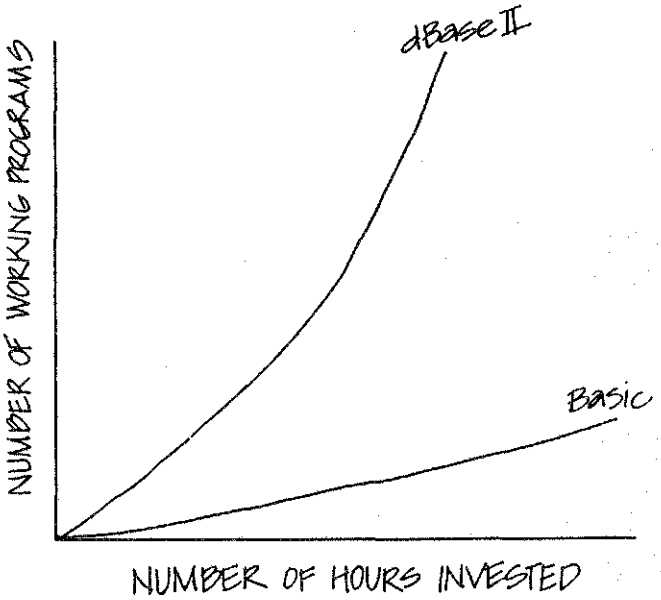
By the way you can even use dBASE to edit your program files. Make sure you are using a diskette with disposable data before first trying out all of these features. Some of the commands will erase your data. Copy all of dBASEII to a experimental diskette and try a

MODIFY COMMAND MAIL

This will let you do a full screen edit of the program. What a feeling to write, edit, run and debug your programs all in one environment. The P-system folks have nothing on us.

The Pages you need the most:
RED SECTION:
SET options p. 84-85
GRAY SECTION:
EDIT keys p.24-25
BROWSE p.47
SET options p. 127-130
COMMAND list p.153

Any software package should also be evaluated in terms of its tutorial value. Too many reviews are pitched directly at professional computer types. dBASEII programming is relatively simple compared to Pascal or advanced BASIC. You could call dBASEII programs giant nested IF's and DO WHILE's, with the Pascal CASE statement thrown in for flavor. If you purcahse dBASEII with the SoftwareBlanc manual you will have a dynamite training course in structured programming. And in the process create some of your much needed business programs. If you take a hard look at your HOURS INVESTED versus WORKING PROGRAMS for the language you use now it tends to be substantial. A graph showing this comparison to dBASEII programming would look like this:



Enough said.

Experiment with some screen input ideas and next time we will compare notes. dBASEII is more than a series of programs, it is an environment for producing fast, enjoyable, working code. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SYSTEM ANALYSIS

conversations with Bruce Huntington

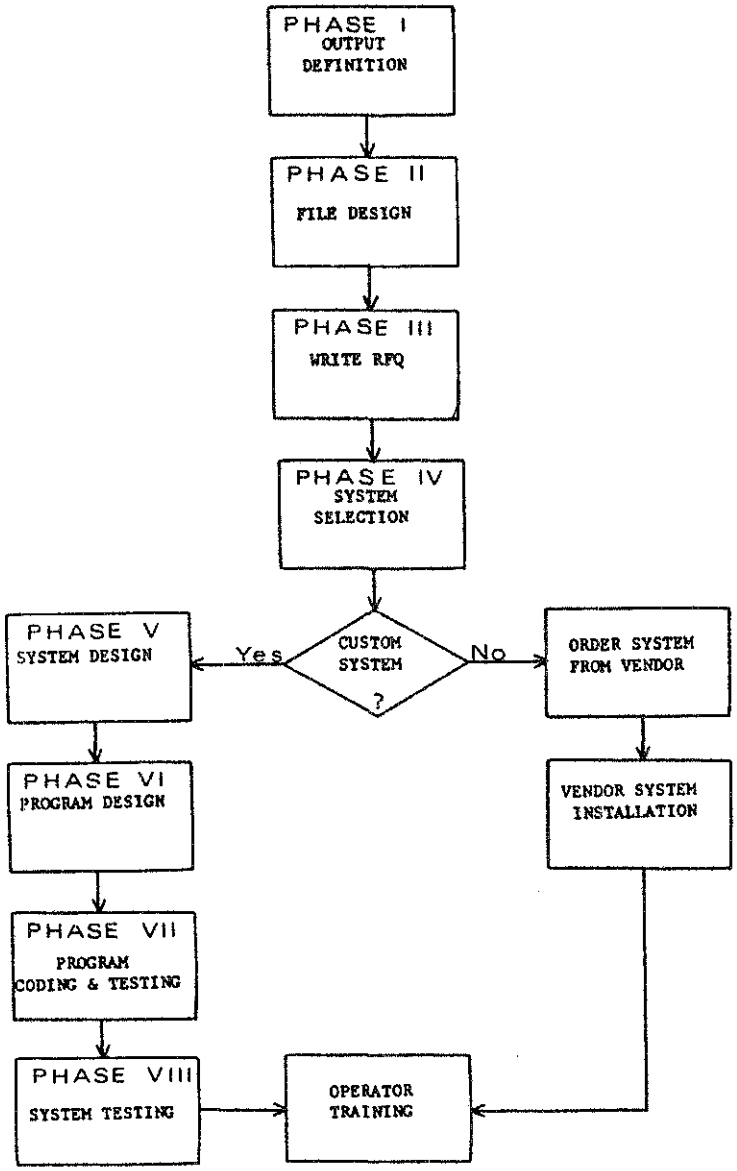
There is a vast difference between writing a program and designing a system, it is the dividing line between a programmer and a programmer analyst.

In most cases when you talk about computerizing something, you are talking about automating a problem that is already being handled in some fashion . Let's suppose a business is trying to keep track of their inventory. Their present method is writing things down on 3x5 cards and throwing them into shoe boxes. No alphabetizing. Anytime they want to find something they dig through the whole box. Now a programmer who knows nothing about analysis will come in and automate that system exactly as it stands. He will produce files that duplicate the shoe boxes and whenever a request is made concerning an inventory item the programs will just search through the files until they collect the information needed. A systems analyst will come in and look at the existing shoe box inventory procedures and before even thinking about computerizing anything he will completely streamlined the current system so it is efficient. From rearranging the stockroom, to establishing inventory check points and indexing categories. And THEN automate it.

If neither the client nor the programmer is experienced at organizing the process in question, then an outside analyst must be called in to efficiently structure the whole inventory system. Even an experienced programmer cannot hope to be an expert at every kind of business system and it is the mark of a professional that he is aware enough to request help in the design stage before he begins the familiar work of computerizing it all for a client. The information flow (what information to deal with, where it is needed, when to use it and why it is necessary) is the responsibility of the client and an analyst, and the analyst may or may not be the programmer depending on his experience.

Once the system has been organized efficiently you can move on to the actual problem of putting it all on computer. Whether you are working for a big firm or programming for yourself the same structure still applies. It is merely more formal if you are working for someone else.

Briefly we are going to show you one possible outline that gets the job done. The diagram illustrates the complete process. We will leave the actual program design steps for another time and just cover phases 1 through 5. In a professional business setting the outline allows the analyst to clearly show where all the money is being spent. The completion of each phase also promotes separating of the work load (and picking up of the paycheck) into manageable pieces. Please note that one of the very LAST phases is picking up a coding pencil and writing the programs. You must start with the screen and printout layouts.



CPS

Phase I OUTPUT DEFINITION

A. For each printed Report, Form and Screen display you must determine the:

1. Actual Layouts
2. Frequency (Daily, Weekly, Monthly, Yearly)
3. Sort Sequence (What order to print them)
4. Subtotals and Totals. (When and where needed.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

B. For each Field used in the Layouts determine

1. The Field Name
2. A Functional Description
3. A List of Possible Values (character types)
4. Locations (Where should the field be printed)
5. Size (For numeric fields allow 1 extra digit)
6. Source Document or Process (Where does this information come from)
7. Creation or Update Frequency (How often does the value change)

C. Review and Verification (with the client)

1. Structured Walk Through
2. Input Review and Check Off
3. Formalization (Signatures)

What you want to do first is to define the input and output. What exactly do you want to get OUT of this computer system.

At the end of PHASE I you have a complete description of your needs expressed in terms of computer outputs. This is a major step in preparing for a meaningful computer system and hardware selection. You have also created the basis for your data dictionary (a description of all the information you want the computer to handle.) From this information a systems analyst can perform PHASE II (File Design.)

PHASE II FILE DESIGN

A. For each File determine:
   1. Fields (Which information belongs in each file)
   2. Record Size (How many fields and in what format)
   3. File Size (How many records will be in each file)
   4. Organization (Indexed, Sequential, etc.)
   5. Storage Type and Size (Hard Disk, tape, etc)

B. Review and Verification

At the end of PHASE II, you have a complete description of all your storage requirements in terms of files.

PHASE III Writing the RFQ (Request for Quotes)

A. Determine specific peripheral requirements (Terminals, Printers, etc.)
B. Determine System Volume Requirements (How many customers, Inventory items, Statements, etc.)
C. Write the RFQ based on the following:
   1. The Outputs (Defined in PHASE I)
   2. The File Requirements (PHASE II)
   3. The Peripheral Requirements
   4. Volume Requirements

D. Review and Verification

PHASE IV System Selection

A. Preliminary vendor Selection
B. Mail our RFQ
C. Criteria Selection (Price, Performance, Peripherals)
D. Review the Bids
E. Prepare an Evaluation Table
F. Vendor Ranking
G. Conclusions and Recommendations
H. Review

If your requirements can be filled by an existing hardware/software system you need only respond to that vendor's quote, otherwise you will need a custom system.

PHASE V

A. Determine System Metabolism
   1. How Will Each file Be Created?
   2. How Will Each File Be Maintained?
   3. What Programs Are Required?
   4. Prepare System Narratives And Documentation.
B. Prepare System Diagram
   1. Formalize File Formats
   2. Formalize The Data Dictionary
   3. Prepare Program Narratives and Documentation
C. Review and Verification

At the end of Phase V you have exhaustive system documentation in the form of narratives and diagrams down to the program level. You have file formats for every file in the system, and a complete and accurate data dictionary defining and describing every field in every file.

From here on out you are dealing with program design and flowcharts. This seems to be the place that many inexperienced programmers start. A well designed system decreases the amount of time spent coding and saves everyone, both client and programmer, the headaches of the "endlessly changing specifications." A problem we have each faced only too often. □

(The previous outline is proprietary information contributed by Computer Programming Services.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ROUNDING

by D. Jones

"Let's make it an even dollar." Rounding, or turning something into a "round" number is something we do quite often in a rough way in our daily affairs. Sometimes we follow special rules : when we buy a half dozen eggs at 75 cents a dozen the checkout clerk always rounds up to 38 cents, but on our taxes we can just drop (truncate) cents to use whole dollars. Usually, though we "round off" to the nearest designated unit. With money items our unit is likely to be the cent. With gasoline it may be tenths of a liter, and so forth.

Finding the nearest unit is simple. If we are a half a unit or less from the next higher number of units then we use the higher number of units. Otherwise we leave our units digit unchanged and drop the extra. For example with cents as our designated unit twenty-five and three-quarters cents rounds off to twenty-six cents while twenty-five and one-quarter cents rounds off to twenty-five cents. Let's look at that in decimal notation.

```
.2575 rounded to 2 decimal places is .26
.2525 rounded to 2 decimal places is .25
```

Another way to state the idea of rounding, in decimal notation, is to look at the digit one to the right of the digit which represents the smallest unit you choose to show your answer. If this number is five or more, add 1 to your designated-unit digit. With our .2575 the smallest unit we wanted to keep was cents, or the second digit to the right of the decimal—the leftmost five. The digit one to the right of that is seven, which is certainly "fiver or more." Following our rule then we add 1 to the five, giving .26. Notice that we could have gotten the same result by adding the following:

```
.2575
.005
----
.2625
```

and chopping off, or truncating the last two digits, giving .26. This is the way rounding is usually implemented on a computer. Here is an example in BASIC:

```
10 rem --------
20 A = 1.03
30 B = A/4+.005
40 PRINT using "##.###";B
```

Without adding the .005 we would get 0.258 instead of the desired 0.263. But be careful because the printing with an edit mask also rounds.

Note: In IBM PC BASIC a rounding takes place automatically when converting from higher precision to lower precision. See example 2 page 3-17 in the IBM BASIC Manual. Do not also round yourself in these situations.

```
WORKING-STORAGE SECTION.
01 DATA-ELEMENTS.
    05 RESULT          PIC 99V99.
    05 EDITED-RESULT   PIC Z.99.

PROCEDURE DIVISION.
    COMPUTE RESULT = 1.030/4  ROUNDED.
    MOVE RESULT TO EDITED-RESULT.
    DISPLAY EDITED-RESULT.
```

With rounding we may make a number larger or smaller, at worst by about half the unit rounded to:

```
.255 is made into .26 which is .005 larger
.254 is made into .25 which is .004 smaller
```

To figure out 1 year's interest charges at 12% on $452.13:

```
10 CHARGE = 452.13*.12+.005
20 PRINT STR$(CHARGE)
    54.2606
```

Calculating by hand, we would have gotten 54.2556. If we would have truncated, we would have gotten 54.25 as the answer. 54.26 is .0044 from the calculated result. .0044 is called the rounding error. If we would have truncated instead, we would have had an error of .0056, a larger error than our rounding error. You may notice that a peculiar asymmetry appears if the amount before rounding has a fixed number of decimal positions. If we round from cents to dollars. We can gain 50 cents at most

—$2.50 can round to $3.00

We can lose a maximum of 49 cents

—$3.49 can round to $3.00

In general, as the number of decimal positions being rounded off becomes large, the possible loss from rounding approaches the possible gain. Also, by the "Law of Large Numbers" , we can expect that in the long run the number of losses from rounding will in most situations be very nearly equal the number of gains. However, the process is still subject to runs of luck—rounding error may be considerable at any given time.

When we are dealing with a sum of several numbers the most accurate answer usually comes from retaining all decimal places during the computation and rounding only the final answer. If instead we rounded each number separately before adding them we accumulate the rounding error. It is possible that adding the intermediate rounding errors together causes them to "cancel out" , but it is also possible that we have a multiplied the amount of error substantially by rounding intermediate results.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We want a result with one decimal place. Rounding the result only

```
A = 1.05
B = 2.05
C = 3.07
--------
  6.17 gives us 6.2
```

Rounding A, B, C, and then adding

```
A rounded = 1.1
B rounded = 2.1
B rounded = 3.1
--------------
This gives  6.3
```

which is not as accurate as the first method. We run into somewhat of a dilemma if we are displaying columns of numbers and a total figure.

Let's say that 300 out of 900 voters vote for candidate A, 300 vote for candidate B, and 300 vote for candidate C, and we want to display percentages. After dividing we get

```
      .333...
   ┌---------
900|300.000
   27000
    30000
    27000     and so on
```

or $.\overline{3}$ in mathematical notation. We have in decimal

	Votes	Percentages
Candidate A	300	33.33
Candidate B	300	33.33
Candidate B	300	33.33
	900	100.00

The percentage column doesn't add up! But it is probably better than having 99.99 appear as the total when in fact 900 votes are 100% of the tally. In general, rounding rules can help you get accurate results.

The number of decimal places your intermediate results carry, whether decimal or binary floating point arithmetic is used, and other aspects of the computing process can influence accuracy. The interested reader can find a good introduction to these topics in "Can you count on your computer?" by Ivars Peterson, Science News, Vol 122, July 31, 1982. Happy rounding and enjoy your IBM Personal Computer. □

Bibliography

1. IBM BASIC Manual, Personal Computer Reference Library
2. Brown, Gary Advanced ANS COBOL With Structured Programming
3. Peterson, Ivars "Can you count on your computer?" Science News, Vol 122, July 31, 1982.

The Norton Utilities™ – DOS Supplement
POWER TOOLS™ for the IBM/PC
14 utilities to enhance IBM/PC – DOS

The essential utilities:
DiskLook – displays everything on diskettes
UnErase – recovers lost files

The advanced utilities:
SecMod – changes any diskette sector
FileHide – controls hidden files

The extra utilities:
TimeMark, Clear, DiskOpt, FileFix
Print, Reverse, Beep, ScrAtr, . . . and more!

Available from computer stores or for $80 from
Peter Norton 1716 Main St. Venice, CA 90291

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMING UTILITIES

Two Cross reference programs.

SUPERCREF $25
The White Ring
5050 Garford # 160
Long Beach, CA 90815

CROSS REFERENCE $24.95
Ensign Software
2312 N.Cole Rd. Suite E
Boise, Idaho 83704

XREF: Cross reference utilities are a standard part of any software development environment. They process your program and supply an alphabetical printout of the variables along with a series of line numbers in which they are found. The cross-reference of a short program looks like this:

```
CES.BAS              SuperCref              01-01-1980
-------------------------------------------------------------------------------


PROGRAM-DEFINED SYMBOLS
-----------------------
#1 ..........  9040
A$ ..........    20    20
BACK ........   200  1020  1070  8093
BS$ .........    30  1130  1140
C ...........  1020  1030  1080  1090  1100  1155  1156  1160
C$ ..........  8905  8905  8910  8910  8910  8910  8920
CC ..........    20
CTLB$ .......    30  1040
ESC$ ........    30  1060
FALSE .......    20  1020
FNCAP$() ....    20  8910
H$ ..........  9030
I ...........  8030  8030  8030  8040  8904  8905  8920  8930
INK$ ........  1040  1040  1040  1050  1050  1060  1100  1130
               1150
MES$ ........    30  1010  1200  1200
OPPS ........  9010  9015  9020
OPPSLINE ....  9010  9020
R ...........  1020  1030  1070  1080  1090  1100  1155  1156
RTN$ ........    30  1100
TRUE ........    20  1070  8093
WD$ .........  1080  1080  1090  1110  1110  1120  1120  8085
               8092  8093  8099  8901  8905  8920
```

Even a listing this simple has a number of important uses. The first service it provides is to show you if you have made any spelling errors. After typing in a name 30 or 40 times you eventually make a mistake and usually correct most of them. When you do misspell a name it can be hard to track down. One place you may write CUSNUM and later on CUNSUM while your eyes tend to blend them together. With a XREF program you have a quick printout of every name you used. Looking at them in alphabetical order you can quickly spot a variable only used once which is misspelled. This effectively solves one difficult debugging problem.

If you have 10 programs and wish to keep accurate records you may print out a final source listing and then a XREF to go with it. Then at any time in the future if you wish to sort through a series of programs locating one variable you have a convenient way to do it.

An additional benefit is for elusive bugs that are caused from incorrect GOSUBs or GOTOs. You can spend hours trying to figure out why an amount has changed value unexpectedly only to find, via a XREF utility, that you entered the subroutine from an inapproprate place.

In the WHITE RING version BASIC keywords are also cross referenced. If you have a COLOR statement that is messing up your palette and you want to go through a large source listing then cross referencing makes it a very simple procedure.

Overall both xref utilities did the job. The companies supplied .BAS versions as well as compiled .EXE versions. The WHITE RING xref considered the fact that you may have both a Color or Mono monitor which is an often neglected feature. The ENSIGN xref program ran a little faster, but did not offer as many options as the WHITE RING version.

When the ENSIGN xref is running your monitor displays the line number it is currently on. The completed printout counts the number of occurances of the variable which the WHITE RING version does not do.

As the WHITE RING xref is working you watch the complete program advancing line by line on your display. The variables cross referenced are highlighted. This takes more time, but is entertaining and lets you know exactly how far the listing has gone. WHITE RING does not give you a count for each occurance, however the printout includes variable names as well as BASIC key words, GOSUBs and GOTOs.

Either one is suitable. There is one feature that both should make a part of their printouts. If you have a large program with a great deal of numeric computations or redefining of strings, you need to know on what line the variable has been changed. Neither xref program does this. The simple placing of an "=" sign before a line number would easily indicate when the variable value has been reset.

The larger your programs are and the more programs you have in one system the greater your need will be for a cross reference utility. You may go for months not bothering to use one, but when the time comes to sort through a big piece of code, they are indispensible. /PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOFTWARE HOUSES

Once you have completed your software package it is time to decide whether to market it yourself or allow a software house to market it for you, This is no easy decision. If you decide to do it yourself you can look forward to ending your peaceful life of a Zen programmer and entering the world of record keeping, packaging, billing, collections, advertising, and that sleeping monster with such an innocent smile called "software support". You might be better off to consider taking a position as "Accommodations Manager" on Noah's Ark, the headaches would probably be less. Whether you have any time left over to program again is questionable, but it certainly depends on the scope of your package and the marketing, financial backing and support it requires. Many of us will take the easy way out and go to a software house. Then the only question is which one?

ROYALTIES

We quickly find out that royalties vary from 15% to 33% (for assembly language programs.) Certainly the bigger the royalty the more money you make, but that depends on the total sales. Obviously receiving 20% from IBM would be better than 25% from Mr. Smiley's Support-ware Inc. Sit down with your copy of VisiCalc and compare the 20% to 25% royalty returns. If one company will pay you 25% and only sells 1,000 copies then you would have been better off going with a firm that pays you only 20% but can sell at least 1,251 copies. The actual royalties a company pays should be very low on your list of priorities for narrowing down your choices. If you have found two companies that are very much alike, then use their percentages as a basis for a final decision. And don't be afraid to bargin. There is an impossible demand for software and when you are faced with a contract manager and he has asked you what percentage you will take, use the old rule of thumb called "pricing according to what the market will bear." And the way you do that is the way a father told his son to sell their horse wagons when he was going to town for the day.

FATHER: I'm be gone for the day son, so just price the wagons according to what the market will bear.

SON: How do I do that?

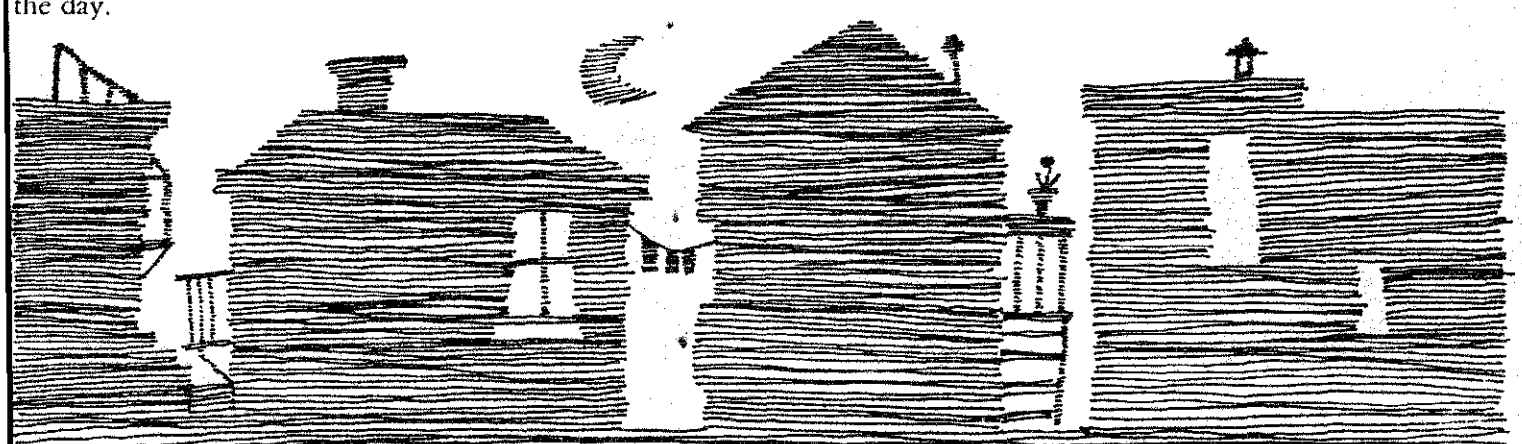
FATHER: Well, when the customer asks what the price of a wagon is you say, "They are $100", and if he doesn't look away, then you say, "And the wheels are $25" and if he doesn't look away, then you say, "A piece."

GAMES V.S. PAYROLL PACKAGES

The kind of software you have written will greatly influence the software house you choose. Most software firms tend to stick to particular varieties of products. Let 's say we have a game and a payroll package we want to market.

For our new version of Space Invaders we want a firm that specializes in games. We do not want a SW house that has a few games as a side line and handles mostly business and accounting packages. Advertising is the key to success. A mediocre program will sell well with good advertising whereas a great program will go nowhere with poor advertising. The competition among games is rather stiff, and good ad copy with dynamite artwork will go a long way toward selling more copies. Look at the ads that the SW house currently has. Are they good? Are they full page? Color? Will your game fit in well? Do they have good financial backing? Do they like your game well enough to give it top billing? Here you must make a decsion based upon intuition. If they consider yours a ho-hum game they may sign a contract with you and place it on a low priority campaign. Going with a newer SW house may be a better bet if they have good financial backing, only 3 games they are selling now, and like your game a lot. Ask to talk with their other authors and find out if they like the firm too.

For our payroll package we want a business SW house. One that specializes in accounting packages because we are going to need support and we don't want to do it. They may also help you with the documentation. With a payroll package it is advisable to find a company that



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

needs a program to fit in with their already existing accounting software. Trying to sell a stand-a-lone product is harder than selling an integrated one. For every complete ACCOUNTING package the company sells, your payroll will go right along with it.

SUPPORT

How much will the SW house help you in the final stages of your project. Some offer up front money (very few do this however.) Others may give or loan you some equipment. Perhaps the greatest aid is writing the documentation for you. The manual that goes with a software package is a tremendous help for future sales. Well written it can save in support costs and increase sales dramatically. Many software products with poorly written manuals leave users cold. If your package needs good documentation and you realize that you need help to write it then this is also a high priority decision.

CONTRACTS

This is a whole topic in itself. You will be faced with numerous decisions. The important one being whether you give the company "exclusive" marketing rights or whether you give them "non-exclusive" marketing rights. Most firms prefer the exclusive kind of contract to prevent you from turning right around and selling the program to another firm or trying to market the package out of your home. Other SW houses will accept non-exclusive terms with the knowledge that their own advertising techniques will dwarf the competition whoever that may be. You should of course copyright your software before you begin sending it around for evaluation. Write for a free kit from the copyright office in Washington D.C. (They have a phone for fast, easy service.)

PAYMENT

After you are on the payroll and the royalties start rolling in you may learn the full extent of what you agreed to. Your 20% percentage is only 20% of a specified figure. This figure is often the wholesale price, minus in house service charges, minus this that and the other thing. Your royalty checks will be broken down several ways. If the SW company sells directly at retail prices then your 20% should be a percentage of the full price minus the exclusions. If they sell only wholsale then it is 20% of the wholesale price. Often they sell both ways and your percentage breakdown should be divided into to retail and wholesale percentages. Request to see a sample payment schedule so you know what to expect.

The firm should also, at your request, make arrangements to have their books audited. This means you may hire a CPA of your choice to go to the company and check up on their record keeping to be sure you are receiving the full royalty amounts and that the number of sold programs is what they say it is.

SUMMARY

Be cautious and check out a company completely before you sign away your software package. The life of a program is several years at best. If you sign a 3 year contract and the SW house does not market your product aggressively during that time then your labor of love will be lost. Take the time to get the phone numbers of other authors working for the firm and talk with them. One phone call may eliminate an otherwise well represented SW company. /PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ASM

Learning assembly language programming on your own is not the easiest of all tasks. In an effort give you a head start on this obstacle our first issue will mention a number of products that should make the transition painless.

To decide whether you want to become a hard core assembly programmer you might consider a "free" assembler that David Whitman is offering. We have not seen a copy of it yet but the price is certainly right. His approach was to write his own rather than spend $100 for the IBM version. This would give you a taste of what it is like with no obligation. See the MISC section of Programmer's Journal for the address of where to write for this assembler.

There are several books on the 8086 instruction set but if your assembly experience is limited you will find them similar to the Pascal manual. Excellent as a resource manual but very poor tutorials. There are 2 books you should start out with, the first one is from INTEL and is called AN INTRODUCTION TO ASM 86. It is the best step by step guide to beginning assembly programming especially for the IBM PC. Our 8088 is merely a slightly modified 8086 chip, so whether a book refers to the 8086 or the 8088 for all practical purposes they are the same. There are just a few instructions that are different on the 8088. You can order this book from INTEL or an IBM PC supplier, Micro Match.

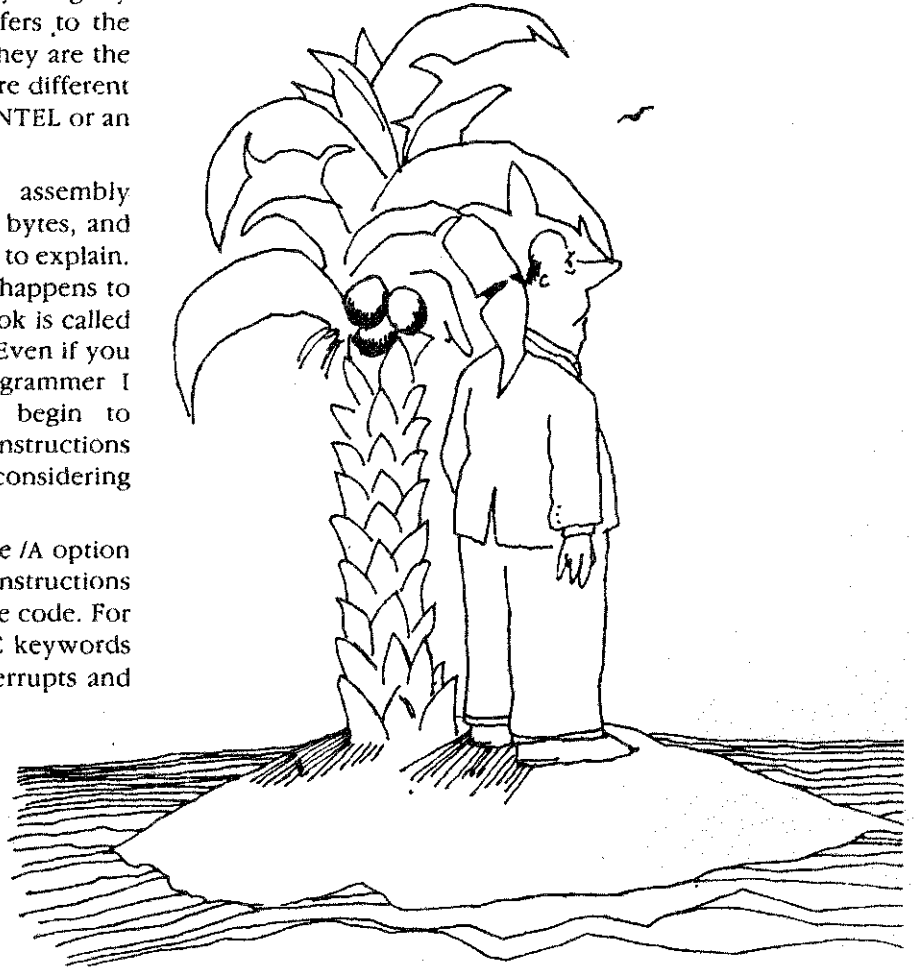
Trying to understand the concept of assembly instructions, why some are 1 byte, others 2 bytes, and how the instruction deals with its data is hard to explain. By far the best explanation I have ever seen happens to also be concerning the IBM PC chip. The book is called THE 8086/8088 PRIMER by Stephen Morse. Even if you do not want to become an assembly programmer I would suggest reading this book. You begin to understand the innerworkings of machine instructions with very little effort. And that is saying a lot considering the complexity of the topic.

If you have a BASIC compiler you can use the /A option to produce object code and understand the instructions generated by each keyword and line of source code. For a real fun time you can run all of the BASIC keywords through with this option and notice the interrupts and assembly code generated.

Testing and debugging assembly code is more difficult because you can easily crash the system. You can imagine the ill will generated in a small shop in which all the programmer are using the same machine and one programmer is doing assembly development. Every time the assembler coder starts testing and kills the system all of the other application programmers may have lost substantial pieces of their source code that they were editing at the time.

With the books listed and a "free" assembler for the asking, your excursion into assembly programming can be relative painless and inexpensive. The more you understand the machine you work on the better your programs will be. /PJ

1. INTEL, AN INTRODUCTION TO ASM86, Intel Corporation,3060 Bowers Ave., Santa Clara, CA 95051, ph.(408) 987-8080
2. Micro Match, book supplier,10343 Commerce Ave., Tujunga, CA 91042 ph.(213) 353-5929
3. Stephen Morse,THE 8086/8088 PRIMER, Hayden Book Company, Inc. Rochelle Park, New Jersey



waiting for updates

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

RESOURCE LIBRARY

BOOT STRAPPING

To bring yourself up to speed in the programming world requires gathering and putting together as much knowledge as possible efficiently. Understanding file handling, screen work, diskette storage, program structure, and so on takes a long time. Much of the frustration can be reduced if you pick the best tutorials along the way. Here are a few of the books and utilities that will speed your progress and make your programming life more meaningful, faster. The learning aids mentioned here are a must if you desire to be computer literate, and even more so as a knowledgeable programmer.

DISKETTES: There is a world of information waiting for you on diskettes. Just being able to LOOK at a file on diskette is an educational experience. The 2 utilities that Peter Norton has written (DISKLOOK and SECMOD) are worth the price 10 times over. DISKLOOK is just what the name implies: you get to look at the information on diskettes in both normal script and in Hex code. Truely a world of information. Seeing the carriage returns and line feeds in a sequential file, or checking out the way the files are allocated on the diskette are very elucidating. An instant tutorial.

TEXT EDITORS

If you are still using EDLIN or the BASIC screen editor to do your text editing you are reducing your programming output by a factor of 2 or 3. The ability to move blocks of code around and not overlay the lines you want to keep is quite frustrating in BASIC. Many is the time you rerun the program only to find the bug is really a lost line or an extra one you forgot to erase. Do yourself an indispensable favor for Christmas and purchase a $100 dollar text editor. The joys are perpetual.

In the mean time you can extract sections of code and write them to disk with the LIST statement. If you want to take 20 lines of code from your current program in BASIC then do a LIST 100-200, "SAVEFILE.BAS" . This will chop out the section you need and write it to disk. Then renumber it and merge it back into the new version.

SOURCE CODE DISKETTES

DISK 1 $15

BASIC INPUT: Includes a professional level input routine that can be plugged into any BASIC program for simple, bug free input. You need only indicate the row, column, and word length. It also provides for prompt messages. The diskette also contains a sample structured program to use as an outline for easy to read modular code.

Mail to: PROGRAMMER'S JOURNAL,

2765 Potter St., Eugene, OR 97405

SOFTWARE TOOLS IN PASCAL by Kernighan and Plauger: This a superb book covering many of the concepts of "good" software. It is written by two experienced and well spoken software designers. They make no bones about praising Pascals good points and mentioning its drawbacks. Whether or not you know Pascal or are even interested in learning it you would do well to read the book. The discussion of Pascal is from the perspective of a novice, even though it goes through the process of developing powerful programming tools before your eyes. Every chapter is filled with advice from truly professional software creators. The book is an essential part of a programmers tool box.

PERIODICALS:

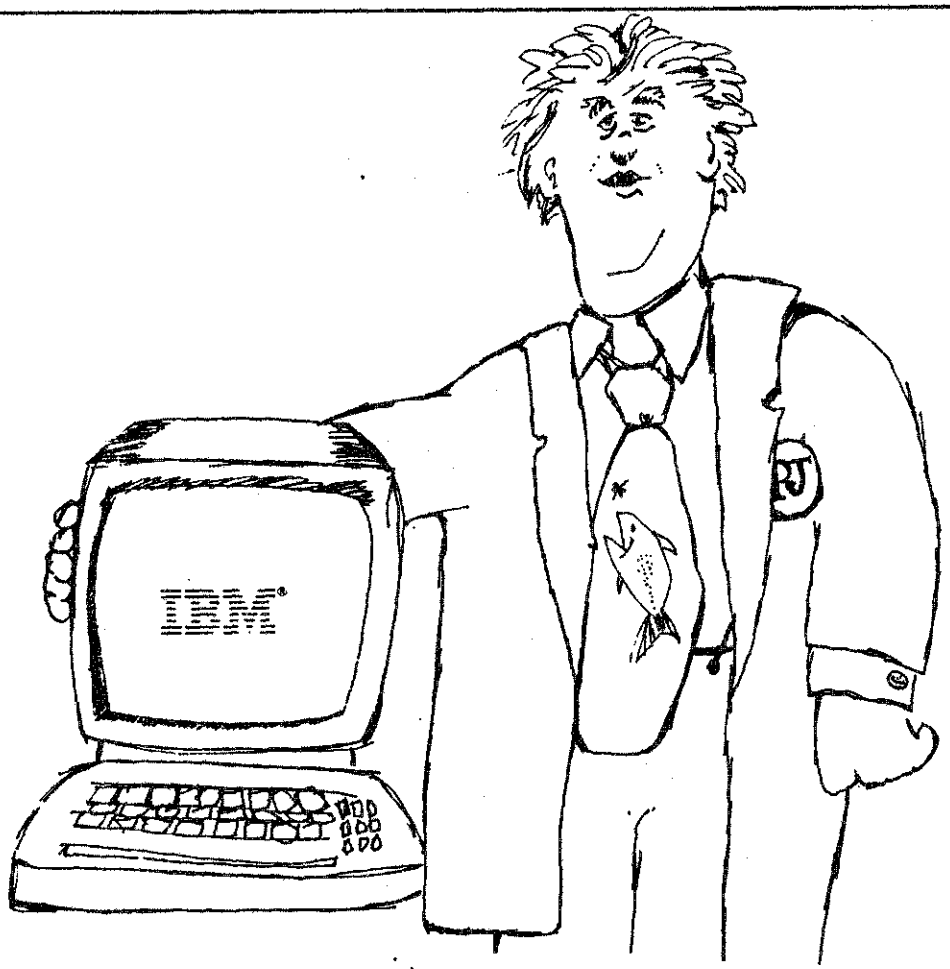
PC: The biggest of the IBM PC magazines. Write PC, 1528 Irving St., San Francisco, CA 94122

SOFTALK: Free to owners of IBM Personal computers. The equivalent of its Apple conterpart. Write SOFTALK for the IBM PC,Box 60, North Holleywood, CA 91601

DR. DOBBS JOURNAL: A journal covering many computers which is starting to have more regular coverage of IBM PC topics. Their July 1982 issue (Number 69) has a "Graphics on the IBM PC" article including assembly language and FORTH listings. You may have to go to a University library for a copy or write directly to Dr. Dobbs,Box E, 1263 El Camino Real, Menlo Park, CA 94025

PERSONAL COMPUTERAGE: A technical magazine with many articles on communications software. Write PERSONAL COMPUTER AGE, P.O. Box 70725, Pasadena, CA 91107

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



RANDOM NOTES:

THE 21st CENTURY: Once in a great while we are thrilled by how far technology has advanced every aspect of our lives. Computers and electronic engineering are leaders of the pack. And if you are wondering how the real hardware geniuses solve their problems, here is a case in point.

We had installed a very large medical computer system and had been having endless problems with the data integrity. We (as the software designers) pointed our fingers at the hardware suppliers and they in turn pointed their fingers at us. Twice a week we would troop out to the installation and run some file maintenance programs to put the data back in some form of decent order while on alternate days the hardware folks would put in a new disk controller or some other suspect component. Finally we narrowed it down to one memory board (everything else had been replaced.) The board supplier reluctantly acknowledged defeat after they put it to the "hair-dryer" test.

Take any memory board with suspect memory chips (not your own, this is state of the art OEM diagnosis) and install it in your computer. Start a memory testing program running. Now in one hand you pick up a hair dryer and in the other you grab some spray coolant. Take the hair dryer and heat up the memory board until you get a repeated memory error, then start spraying the coolant on one chip at a time until the memory error stops. Waa-laa, you have found the defective memory chip. State of the art in every way. I wonder if they had it on "Fluff" or "Dry" ?

ROM: How many programmers does it take to change a light bulb?

RAM: None, that's a HARDWARE problem !

ERROR MESSAGE OF THE MONTH:

Imagine your program just bombed out with an error code 35 and you went to your manual to find out what the problem was and how to fix it. The following message is from an honest to goodness real live computer manual.

ERROR CODE 35

Reason: An accept input operation was issued to a nonnever-ending program that had no outstanding invite input operations.

Recovery: Review program logic. An accept input is valid for a nonnever-ending program only if one or more invite inputs are outstanding.

—submitted by Lawrence Wilson, Norman, Oklahoma

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Free RatBAS;

If you want an excellent tutorial on structured BASIC and a free translator to allow you to write more Pascal like BASIC code then send for a copy of the RatBAS translator program with documentation. Send self-addresses stamped mailer with a formated diskette to

Brent Weaver
Wells Fargo Inv. Advisors
P.O. Box 44029
San Francisco, CA 94144

OPEN LETTER TO GRAPHICS PROGRAMMERS:

If your program uses color graphics then every program should begin with the following lines:

```
10 rem switch to color absolutely
20 DEF SEG
30 POKE &H410,(PEEK(&H410) AND &HCF) OR &H10
40 SCREEN 1,0,0,0
50 SCREEN 0
60 WIDTH 40
70 LOCATE ,,1,6,7
```

See appendix I-8 of the newer BASIC manual. The new manual has a lot of additional information on using BASIC even though many stores indicate otherwise.

There are a number of IBM PC setups
1. Color adapter with a T.V.
2. Color adapter with an RGB monitor
3. Color adapter with Mono and Color.

Your programs should be tested on each of these system setups. A T.V. has some problems with PSET lines and they may not appear correctly.

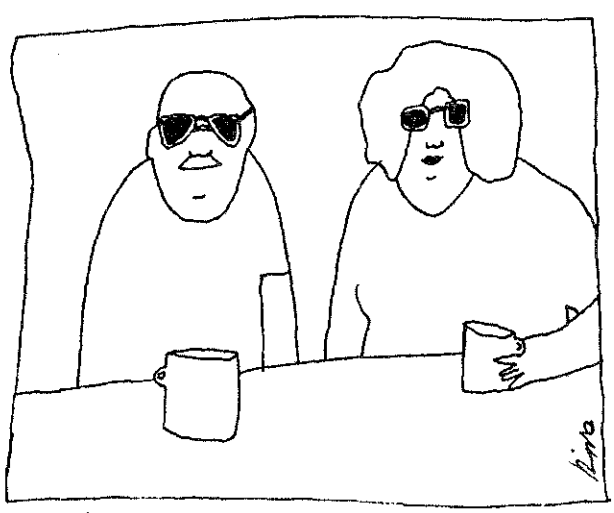
But most important, a certain number of users have both a Mono and Color adapter setup and if you fail to begin EACH program with the preceeding POKE lines then the program will default to Mono and the user will be stuck with garbage on a green screen. A double monitor system always defaults back to Mono when DOS is called via the SYSTEM statement.

I repeat, EVERY color program should begin with the above lines to ensure a switch to COLOR. It is a nice option (example the Microcosm game) to check and see if they have both monitors, and if they do ask them if they wish to switch assuming your program will run on both.

Free Assembler:

David Whitman is offering a "free" assembler for those of you interested in assembly language programming but do not want to spend $100 for the IBM version to see if you like it or not. If you like and use the program he requests (like PC-Talk) a $20 donation. The assembler is called CHASM. Send a diskette with return postage and mailer to:

David Whitman
2 North Park St. Apt L
Hanover, NH 03755



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S JOURNAL

The Resource Newsletter for IBM PC Programmers

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

1 year subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature _____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

Languages you use: _____

Languages interested in: _____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PROGRAMMER'S JOURNAL

The Resource Newsletter for IBM PC Programmers

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

1 year subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature _____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

Languages you use: _____

Languages interested in: _____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMING UP NEXT . . .

Is it my Hardware or my Software?

How to decide what's at fault when you have a system failure. Tips on narrowing down the possibilities.

A BASIC Input Routine

Plug it into your programs and enjoy bug free input with the greatest of ease.

Structured Code

What should modular/structured code REALLY look like.

Character Generators

Create your own graphic shapes to use with games, creating type faces, foreign language letters, etc.

"C"

A review of Small "C" and what you should look for when choosing a "C" compiler.

Software Submission Procedures

Look professional when you send in your programs.

Assembly Source Code

Some hard core assembly lines.

dBASEII

How to create your dBASE screen formats that maximize user friendly input.

Graphic Tricks for Writing Games

It doesn't all have to be in machine code.

DOS BOSS, PJ Profile, Pascal, BASIC Compiler aids, Software Reviews, and a whole lot more!

$4.00

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED